THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par
Université Toulouse III - Paul Sabatier
Discipline ou spécialité :
Informatique

Présentée et soutenue par
Jérémy Boes
Le
28 mars 2014
Titre :
Apprentissage du contrôle de systèmes complexes
par l'auto-organisation coopérative d'un système multi-agent :
application à la calibration de moteurs à combustion

JURY
Vincent Chevrier, Maître de conférences, Université de Lorraine (rapporteur)
Marie-Pierre Gleizes, Professeur, Université Paul Sabatier – Toulouse III (présidente)
Pierre Glize, Ingénieur HDR CNRS (directeur)
Pascal Higelin, Professeur, Université d'Orléans (examinateur)
Vincent Huc – Ingénieur, Aboard Engineering (invité)
Frédéric Migeon, Maître de conférences, Université Paul Sabatier – Toulouse III (encadrant)
Michel Occello, Professeur, Université Pierre Mendès France - Grenoble II (rapporteur)

Ecole doctorale : Mathématiques Informatique Télécommunications (MITT)
Unité de recherche : Institut de Recherche en Informatique de Toulouse (IRIT)
Directeur(s) de Thèse : Pierre Glize et Frédéric Migeon
Rapporteurs : Vincent Chevrier et Michel Occello

Résumé

Cette thèse s'intéresse au contrôle de systèmes complexes, et propose une solution multi-agent. Contrôler un système, c'est appliquer les modifications adéquates sur ses entrées de façon à placer ses sorties dans un état attendu. Les méthodes habituelles se basent majoritairement sur l'utilisation de modèles mathématiques du système contrôlé, afin de calculer les actions de contrôle à effectuer. Ces méthodes trouvent leurs limites face aux systèmes complexes, qui ont une dynamique non-linéaire, et sont souvent bruités et instables. La construction d'un modèle est dans ce cas une tâche ardue, qui peut s'étendre sur plusieurs années. La plupart des méthodes proposent alors d'utiliser un algorithme d'apprentissage artificiel pour apprendre un modèle. Cependant, le modèle produit demeure difficile à exploiter pour le contrôle, puisqu'il reproduit les caractéristiques difficiles du système réel, notamment sa non-linéarité. Une meilleure approche, adoptée dans cette thèse, consiste à apprendre directement le contrôle. La loi de la variété requise indique que, pour être capable d'accomplir sa tâche, le contrôleur doit être au moins aussi complexe que le système contrôlé. Il faut donc concevoir un système capable d'apprendre, de contrôler, et surtout, de franchir le mur de la complexité.

La distribution du contrôle, c'est-à-dire l'affectation du contrôle de chaque entrée d'un système à des contrôleurs plus ou moins indépendants, permet de s'attaquer à la complexité. Mais cela demeure un sujet de recherche actif, à plus forte raison lorsque vient s'ajouter une problématique d'apprentissage. Les systèmes multi-agents (SMA), composés d'entités autonomes, se prêtent naturellement aux problèmes distribués et peuvent ainsi beaucoup apporter. En particulier, les systèmes multi-agents adaptatifs (AMAS) s'appuient sur l'auto-organisation des agents pour faire émerger une fonction globale adéquate. Cette auto-organisation est guidée par la coopération. Chaque agent est capable de détecter et de résoudre les situations dans lesquelles il ne peut accomplir sa tâche. Un AMAS est ainsi doté de fortes capacités d'adaptation et d'apprentissage. Il est également capable, grâce à l'émergence, d'accomplir des tâches complexes. Appliquée au problème du contrôle et de son apprentissage, cette approche conduit à la définition d'un SMA particulier, présenté dans cette thèse. Les expérimentations, menées sur des simulations ainsi qu'en situation réelle (sur un moteur à combustion), ont montré la capacité du système à apprendre le contrôle de plusieurs entrées en fonction de critères sur plusieurs sorties, tout en étant robuste aux perturbations, et facile à instancier. Ces résultats sont analysés pour conclure sur la validité du système.

Remerciements

Jusqu'au bout, j'ai cru que cette partie serait la plus facile à écrire. Je me trompais. Devant l'inéluctabilité de la rédaction désormais accomplie de ce document, j'ai parfois le sentiment que toutes les interactions que j'ai connues depuis ma naissance sont responsables de cet aboutissement. Cela est possiblement vrai (et incomplet), mais implique alors que toutes les personnes avec lesquelles je suis entré en contact durant ma vie devraient être remerciées ici. L'exercice perdrait alors tout son intérêt. Je vais donc me limiter à celles qui ont eu un impact direct sur cette thèse, et sur la manière dont je l'ai abordée. Tâchant de n'oublier personne, je vais naviguer le long de la chaîne de causalités qui a débouché sur les lignes que vous lisez actuellement.

L'influence la plus récente est celle des rapporteurs. Je remercie donc Vincent Chevrier et Michel Occello, pour avoir pris le temps de lire et commenter cette thèse. À l'heure où ses lignes sont écrites, ils s'apprêtent à gagner Toulouse en tant que membres du jury, où ils seront rejoints par Pascal Higelin, que je remercie également.

Viennent ensuite mes encadrants. Merci Pierre Glize, pour m'avoir fait découvrir les AMAS avec un enthousiasme communicatif, pour les quelques échanges nocturnes de mails philosophico-scientifiques, les quelques discussions diurnes qui ont suivi, et pour ton optimisme réconfortant. Travailler à tes côtés va me manquer. Merci Frédéric Migeon, pour les nombreux conseils allant au-delà du travail, pour les madeleines, et la franchise qui te caractérise. C'est parce que j'appréciais ta manière d'être en TD que j'ai choisi un sujet que tu encadrais, et je ne regrette pas cette décision.

Si j'ai passé un peu plus de trois excellentes années de thèse, c'est aussi grâce à l'équipe SMAC. Pour cela, je dois remercier Marie-Pierre Gleizes, pour toute l'énergie qu'elle met dans l'équipe, pour les repas annuels et les séminaires. Ces moments ont grandement contribué aux liens tissés avec les différents membres de l'équipe, et donc au bien-être quotidien sur le lieu de travail. Les smackers sont trop nombreux pour être tous nommément cités, aussi vais-je me limiter à ceux avec lesquels mes interactions ont été les plus fréquentes. Tout d'abord, merci à mon co-bureau le plus récent, Charles Triboulot, pour tous les moments de détente. Je te remercie également d'être quelqu'un de silencieux, contrairement à mon premier co-bureau, Victor Noël. Ce dernier est probablement celui qui a eu le plus d'impact sur ma thèse, en dehors de mes encadrants. Sans lui, j'aurais terminé deux fois plus vite (pour un résultat probablement deux fois moins bien), alors je suis bien obligé de le remercier. Et il faut bien avouer que ses conseils ont toujours été pertinents, et nos échanges, pour moi,

profondément enrichissants. Par contre, je ne remercie pas Valérian Guivarch de s'être incrusté au 359, mais je reconnais que son ouverture d'esprit est agréable à côtoyer. Önder Gürcan a également partagé cet espace par intermittence, et fut toujours d'excellente compagnie, merci à lui. Merci à Nicolas Brax pour ShadowRun, à François Gatto pour ne m'avoir pas laissé tout seul lors des longues soirées de rédaction à l'IRIT, et à tous les (ex-)doctorants du 3ème étage et visiteurs occasionnels, Luc Pons, Arcady Rantrua, Grégoire Denis, Noelie Bonjean, Célia Picard, Tom Jorquera, Simon Stuker, Sylvain Lemouzy, Raja Boulbel, Zeineb Graja, Sylvain Videau, Elsy Kaddoum, Teddy Bouziat, Alexandre Perles, Clément Mignard, Bob, et Julien Martin. Si les pauses cafés sont chouettes, c'est grâce à vous (et tant pis si le couloir est bruyant). Merci à Kévin Serin pour son excellent travail sur les interfaces graphiques d'ESCHER, à Éric Andonoff pour les discussions montagne toujours rafraîchissantes, à Frédéric Amblard pour son humour, et à tous les autres que je ne peux citer ici. Pour en terminer avec SMAC, un dernier merci tout particulier à Christine Maurel. Je vous appréciais déjà en tant qu'enseignante, mais l'attention que vous portez à vos "petits", dont je me suis plu à faire partie, a été un soutien au moral de première importance. Votez Lambda !

Pour continuer sur la chaîne de causalités, je remercie Aboard Engineering pour tout le travail effectué avec eux. Merci en particulier à Vincent Huc et Paul Martin, ainsi qu'à Erwan Salvy pour sa très précieuse aide au banc d'essai.

La chaîne nous emmène maintenant hors de la sphère du travail pour en venir aux amis rencontrés à Toulouse. Rémi Abbal est la première personne avec qui j'ai sympathisé sur les bancs de l'Université Paul Sabatier. Ses notes de cours, sa bonne humeur continuelle ainsi que son écoute m'ont été précieuses, alors merci ! Merci Arnaud Oglaza pour avoir dégoté de bons bars toulousains, m'avoir fait découvrir le rugby, et pour ton esprit de compétition toujours stimulant. Merci Raphaël Hoarau pour INS/MV, pour la bête sauvage, et pour cette touche unique et indispensable que tu apportes à ce groupe d'amis. Merci Jésus pour ce que tu sais et dont je ne peux décemment parler ici, merci Maxime Cordeil, Yannick Gimenez, Louis Noval, Maxime Reynal et tous les autres pour les bons moments passés ensemble.

D'autres personnes ont eu un impact moins direct, mais tout aussi important. Je tiens à remercier Romain Corradini, pour avoir, sans le vouloir, forgé mon caractère. Tous mes autres amis se reconnaîtront dans ce remerciement qui leur est également adressé.

Merci Maroussia. C'est parce que tu es là que j'ai pu faire ce bout de chemin. Une thèse entière ne suffirait pas à raconter tout ce que tu m'apportes.

Merci Papa (et Christine), merci Maman (et Bruno), d'être des modèles dont je suis fier. Pour la même raison, merci Pierre, Marine, et Lola, et désolé de n'avoir pas été très présent ces dernières années. Merci Thérèse et Gaby, cette dernière remarque s'adresse aussi à vous. Merci cousins/cousines, tatas/tontons, qui êtes bien trop nombreux pour être cités. Merci Pépé, merci Mémé, merci Mamie, je puise en vous courage et inspiration depuis bien longtemps.

Et pour finir, je te remercie, toi lecteur qui s'apprête à tourner les pages. S'il y a quelqu'un pour qui ce document a été écrit, c'est bien toi !

Table des matières

Introduction

Que se passe-t-il lorsqu'un enfant apprend, pour la première fois, à faire du vélo ? Un adulte l'installe sur la selle, c'est l'initialisation. À ce moment, l'enfant n'a aucune connaissance en physique. Masse, inertie, forces de frottements, poussée, portance, accélération gravitationnelle, ou encore énergie cinétique, sont autant de concepts qui lui sont inconnus. Il est bien incapable de calculer, au sens mathématique du terme, quelles forces appliquer sur les pédales et le guidon afin de mener son vélo là où il l'entend. Malgré tout, l'enfant va essayer quelques mouvements, quelques actions qui vont avoir des effets sur le comportement du vélo. Petit à petit, et probablement au prix d'une chute ou deux, l'enfant va parvenir à contrôler son vélo. Il a appris comment utiliser les points de contrôle à sa disposition pour avancer, freiner, tourner, etc. L'enfant maîtrise maintenant son vélo, il peut pleinement profiter de la liberté qu'offre ce moyen de transport. Et pourtant il ne connaît toujours rien aux théories de la dynamique et de la mécanique du solide. Les processus physiques en jeu au sein de sa bicyclette lui demeurent étrangers. Il contrôle un système sans en avoir construit un modèle analytique décrivant son comportement. Tout vient de l'expérience. L'enfant reconnaît des situations qu'il a déjà rencontrées, et sait quelles sont les actions à faire (ou à ne pas faire !) pour garder le contrôle du vélo. Voilà qui résume, de manière imagée, le point de vue adopté par cette thèse au sujet du contrôle de systèmes complexes.

Objectif de la thèse

Contrôler un système, c'est être capable d'effectuer les manipulations adéquates sur ses entrées afin d'en placer les sorties dans un état désiré. Le contrôle est un domaine situé au carrefour de l'automatique, des mathématiques, de l'informatique et de l'ingénierie. La tendance actuelle, en particulier dans l'industrie, est de commencer par construire un modèle mathématique précis du système à contrôler, puis de s'en servir pour calculer les actions à entreprendre en fonction des consignes.

Devant la difficulté et le coût de l'élaboration (puis du paramétrage) d'un modèle, une alternative souvent adoptée est de l'apprendre. Doter un contrôleur de capacités d'apprentissage, afin qu'il se construise lui-même un modèle du système contrôlé pour ensuite l'exploiter et calculer les actions à entreprendre, est une solution séduisante. Cependant, elle montre ses limites sur les systèmes complexes. En effet, l'utilisation qui est faite du modèle est tributaire de la complexité de celui-ci. La complexité d'un modèle, notamment la non-linéarité, rend

rapidement le calcul du contrôle bien trop coûteux, voire tout bonnement impossible. Il existe une autre possibilité : apprendre non pas un modèle du système contrôlé, mais le contrôle lui-même. La difficulté est ainsi contournée, puisque si la complexité du contrôle demeure, celle du système contrôlé est en quelque sorte masquée. On ne se concentre que sur ses entrées et sorties, sans chercher à expliciter son fonctionnement interne.

L'objectif de cette thèse est donc de concevoir un système capable d'apprendre à contrôler, sans posséder de connaissances préalables sur le système auquel il s'applique. L'apprentissage doit se faire en temps réel, à partir de l'observation des entrées et des sorties du système contrôlé.

L'intérêt du contrôle de moteurs

Le système de contrôle présenté dans cette thèse se veut générique. Néanmoins, disposer d'un terrain d'expérimentation concret permet de se confronter à des problèmes réels. Cela est donc primordial dans le cadre de travaux sur le contrôle. Pour cette thèse, il s'agit des moteurs à combustion.

Au cours des dernières années, les moteurs à combustion interne utilisés par l'industrie automobile se sont significativement complexifiés. On note en particulier les arrivées de nouvelles technologies comme les filtres à particules, ou encore les systèmes EGR (*Exhaust Gas Recirculation*) permettant de recycler les gaz d'échappement. Ces innovations améliorent le rendement des moteurs, mais, en contrepartie, elles en complexifient d'autant plus le contrôle. En outre, les exigences sur les performances des moteurs se sont accrues, principalement en raison du renforcement des normes antipollution. Dans ces conditions, le contrôle prend une place cruciale. En effet, la manipulation adéquate des paramètres d'entrée d'un moteur est garante de la satisfaction de ces exigences.

L'électronique et le numérique, maintenant bien implantés dans le domaine de l'automobile, permettent la mise en œuvre des techniques avancées de contrôle, issues de l'informatique (citons par exemple la commande prédictive, basée sur l'utilisation de modèles). Un moteur étant un système dynamique, non-linéaire, bruité et instable, il représente un défi d'envergure quant à l'apprentissage de son contrôle. Les moteurs à combustion constituent donc un domaine d'application très intéressant.

Un projet multidisciplinaire

La plus grande partie des travaux présentés dans ce document se sont déroulés au sein du projet ORIANNE (Outil numéRIque pour le mAquettage de foNctions de coNtrôle motEur). Ce projet, soutenu par le Fond Unique Interministériel, a pour but d'élaborer un outil permettant le prototypage rapide d'un contrôleur de moteur. Il inclut des problématiques allant du matériel électronique utilisé pour le calculateur à la conception de son logiciel. Aussi, il implique de nombreux partenaires, industriels comme académiques, provenant d'horizons variés :

- Aboard Engineering, bureau d'études en automatique, électronique et informatique industrielle, et leader du projet ;
- FH Electronics, concepteur de calculateurs électroniques embarqués ;
- Renault, constructeur automobile ;
- le Centre d'Étude et de Recherche Technologique en Aérothermique et Moteurs (CERTAM) ;
- le Centre d'Études Vibro-Acoustiques pour l'Automobile (CEVAA) ;
- l'Institut de Recherche en Systèmes Électroniques Embarqués (IRSEEM) ;
- l'Institut de Recherche en Informatique de Toulouse (IRIT), représenté par l'équipe de recherche Systèmes Multi-Agents Coopératifs (SMAC).

La tâche dédiée à l'IRIT concerne la conception d'un outil de calibration automatique, c'est-à-dire un système capable d'apprendre le paramétrage optimal du logiciel d'un calculateur. Cette tâche s'apparente en fait à l'apprentissage du contrôle du moteur, et a constitué l'activité centrale de cette thèse.

Organisation du document

Le premier chapitre de ce document précise le contexte de nos travaux. Il trace un bref historique du contrôle de systèmes, en aborde les notions élémentaires, et présente le cas particulier du contrôle de moteurs. Il met le doigt sur la nécessité d'apprendre le contrôle.

Le second chapitre est un état de l'art des méthodes de contrôle, ainsi que des techniques d'apprentissage artificiel. Il présente un large éventail d'approches, et en étudie les manques. Il souligne notamment la difficulté que représente l'instanciation des méthodes existantes à un cas concret.

C'est dans le chapitre 3 que sont introduits les systèmes multi-agents. Cette branche particulière de l'intelligence artificielle présente de nombreuses qualités intéressantes vis-à-vis des problèmes posés par le contrôle de systèmes et son apprentissage. Les notions d'auto-organisation et d'émergence sont abordées, puis la théorie des systèmes multi-agents adaptatifs (*Adaptive Multi-Agent Systems*, AMAS) est exposée. Cette approche place la notion de coopération au centre du comportement des agents, et permet de concevoir des systèmes auto-organisateurs dotés d'une grande capacité d'adaptation et d'apprentissage.

Le chapitre 4 présente et analyse ESCHER (*Emergent Self-adaptive Controller for Heat Engine calibRation*), un système multi-agent adaptatif capable d'apprendre en temps réel le contrôle d'un système complexe. Conçu et développé durant cette thèse, il en constitue la principale contribution.

En réponse au problème récurrent de l'indisponibilité de certaines ressources critiques dans un projet (en l'occurrence celle d'un moteur et de son banc d'essai), ainsi que pour assurer la généricité de ESCHER, ce dernier a été testé sur des "boîtes noires" tout au long de son développement. Ces boîtes ont été générées par BACH (*Builder of Abstract maCHines*), un autre système multi-agent adaptatif conçu et développé lors de cette thèse. Il est présenté dans le chapitre 5.

Enfin, le chapitre 6 montre les expérimentations conduites sur des boîtes noires générées, ainsi que sur un moteur réel. Les résultats obtenus aboutissent à la validation des deux systèmes, et à la caractérisation de leurs limites.

CHAPITRE 1

Contexte de la thèse

L'objectif de ce chapitre est d'introduire la problématique générale du contrôle de systèmes complexes, et de préciser les objectifs de cette thèse. La première partie du chapitre s'intéresse au contrôle de manière générale, esquisse un rapide historique et pose les principales définitions, tandis que la deuxième partie se concentre sur le contrôle de moteurs à combustion interne.

1.1 Le contrôle de systèmes complexes

Le contrôle est une notion très générique qui s'exprime de bien des manières. Cette section consiste en une présentation générale du contrôle de systèmes, de son histoire et des principaux concepts manipulés.

1.1.1 Historique

Depuis toujours, les humains cherchent à agir sur l'environnement afin d'améliorer leur condition. Trouver quelles sont les actions à entreprendre pour arriver à ses fins est l'essence même du contrôle. On parle plus précisément de contrôle de systèmes lorsque ces actions sont effectuées sur les entrées d'un système particulier. Avant d'être une science théorisée, le contrôle relevait purement de l'ingénierie car il est fortement ancré dans les problèmes pratiques.

1.1.1.1 La clepsydre de Ctésibios

Un des exemples les plus connus nous est donné par le problème de la mesure du temps dans l'Antiquité. La clepsydre égyptienne est en effet un des premiers systèmes sur lequel fut appliqué un raisonnement de contrôle afin d'en améliorer le fonctionnement. Une clepsydre égyptienne consiste en un réservoir, rempli d'eau, dont la base est trouée. L'eau s'écoule progressivement par ce trou et est réceptionnée dans un deuxième récipient. Celui-ci est

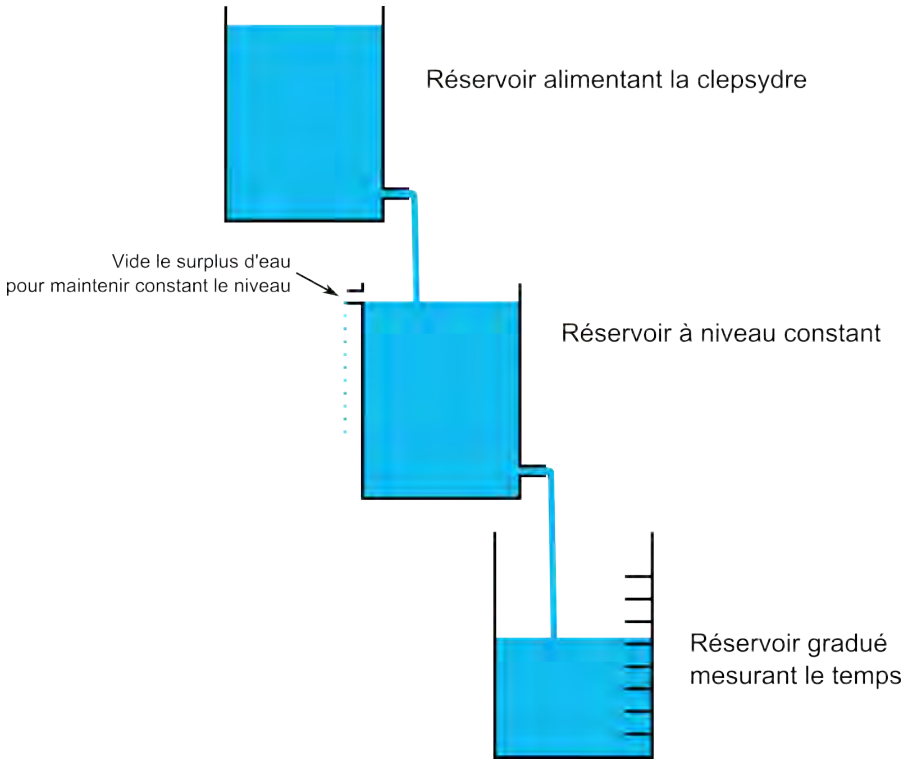

Figure 1.1 – Schéma de la clepsydre grecque de Ctésibios.

gradué, de sorte que le niveau d'eau indique une mesure du temps passé. Cependant le volume d'eau du premier seau diminuant petit à petit, la pression exercée devient de moins en moins importante. Cela cause une variation du débit entre les deux récipients qui entraîne une approximation de plus en plus importante des mesures. C'est pour remédier à ce problème que le grec Ctésibios d'Alexandrie proposa, au III^{e} siècle avant J-C, d'ajouter un troisième seau (Ronan 1983). Celui-ci est placé entre les deux récipients originaux, et son niveau d'eau est maintenu constant à l'aide d'un mécanisme de rejet du surplus d'eau (figure 1.1). Ainsi le débit entrant du vase gradué ne varie plus et les mesures sont plus précises. Révolutionnaire à l'époque, le même principe est encore utilisé aujourd'hui, par exemple dans les carburateurs de voiture.

Cet exemple de système de contrôle automatique illustre la volonté d'alimenter de manière adéquate un système afin que celui-ci donne le résultat attendu. Dans le cas de la clepsydre de Ctésibios, il s'agit d'une forme particulière du contrôle, la régulation, qui consiste à s'assurer de la stabilité de l'activité d'un système. Mais le contrôle consiste aussi (et surtout) à être capable de faire varier la sortie d'un système selon les souhaits de l'utilisateur.

1.1.1.2 De la révolution industrielle à nos jours

C'est au XIX^{e} siècle, lors de la révolution industrielle, que l'utilisation massive de machines donna aux systèmes de contrôle automatique une importance cruciale. Réguler température,

pression, et autres niveaux ne pouvait effectivement pas se faire manuellement sur des systèmes tels que les machines à vapeur. Toutefois, les contrôleurs de l'époque manquaient encore de recul théorique et étaient majoritairement élaborés à partir de l'expérience et de l'intuition des ingénieurs.

Quelques percées mathématiques eurent tout de même lieu et conduisirent plus tard à la théorie du contrôle. Par exemple, le mathématicien et astronome George Biddell Airy utilisa en 1840 des équations différentielles pour caractériser l'instabilité d'un système de contrôle de télescope. La publication en 1868 d'un article de James Clerk Maxwell sur les régulateurs à boules (un système mécanique de régulation de la vitesse de rotation des machines à vapeur) est aujourd'hui considérée comme le début de la théorie du contrôle.

Jusqu'au début du XX^{e} siècle, la notion de système était absente. Les concepts d'entrée et de sortie firent leur apparition à cette époque avec l'arrivée de la théorie des systèmes. Ce siècle fut marqué par les deux guerres mondiales ainsi que par l'essor des télécommunications. Les guerres appuyèrent les recherches sur le guidage d'engins et sur la visée de missiles. Les contrôleurs de type PID, une technique toujours d'actualité (présentée dans le chapitre suivant), furent, notamment, utilisés pour la première fois en 1911 pour le contrôle de la direction de navires, avant d'être analysés théoriquement en 1922 (Minorsky 1922).

Les technologies de communication à distance firent quant à elles naître des problématiques d'amplification du signal et donnèrent lieu à la notion de rétroaction négative, découverte par Harold Stephen Black en 1927. Elles entraînèrent également l'application des travaux de mathématiciens du siècle précédent, comme Augustin-Louis Cauchy ou Joseph Fourier, ouvrant la voie du contrôle par domaine fréquentiel.

Les travaux publiés jusqu'à cette époque constituent ce que l'on appelle aujourd'hui la théorie classique du contrôle. Suffisante dans beaucoup de cas, cette théorie est néanmoins en difficulté face aux systèmes non-linéaires et possédant plusieurs entrées et sorties. L'intérêt se porta alors sur le contrôle optimal, qui vise à réduire une certaine mesure de coût de fonctionnement d'un système, et sur l'utilisation d'équations différentielles pour analyser les systèmes et leur contrôle. C'est à ce moment que la discipline du contrôle rencontra celle de l'informatique, alors en plein développement. La faculté de cette dernière à calculer les solutions d'équations complexes a favorisé la mise en équation des systèmes de contrôle, ainsi que la modélisation mathématique des systèmes contrôlés. Cette tendance se poursuit depuis les années 70 et est progressivement complétée par l'utilisation de techniques issues de l'intelligence artificielle, en particulier les algorithmes d'apprentissage.

Ce bref historique appuie sur le lien entre la théorie du contrôle et les besoins représentatifs des époques et des technologies dans lesquelles elle s'inscrit. Les méthodes de contrôle, créées bien souvent par des spécialistes du domaine d'application, ont des objectifs variés (régulation, optimisation, maîtrise, etc) qui trouvent tous une solution en la manipulation adéquate des entrées d'un système.

La section suivante expose les concepts fondamentaux utiles pour aborder le problème du contrôle.

1.1.2 Définitions et concepts fondamentaux

Le contrôle de systèmes est au carrefour de plusieurs disciplines : automatique, mathématique et aujourd'hui informatique, et s'étend à de nombreux domaines d'application (biologie, physique, chimie, etc) apportant chacun leur vision du problème, leurs exigences et leurs contraintes. L'étude des systèmes et de leur contrôle met tout de même à jours des traits communs, faisant apparaître des concepts propres au contrôle, présentés dans cette section.

1.1.2.1 La notion de système

La notion de contrôle ne saurait se départir de celle de système. Cette dernière prend différentes formes selon la discipline concernée. Mathématiques, informatique, physique ou encore automatique considèrent un "système" sous différents angles. Deux aspects complémentaires peuvent être mis en avant. Un système peut être décrit selon son aspect structurel. Il est alors présenté comme un ensemble d'éléments constitutifs, leurs relations et par une frontière déterminant l'interface entre le système et son environnement. Mais un système peut également être abordé selon son aspect fonctionnel. On s'intéresse alors aux flux d'information. L'accent est mis sur les entrées et les sorties (du système ou de ses composants) et sur le processus de transformation qui lie les premières aux secondes.

Le contrôle privilégie généralement ce deuxième point de vue. Les sorties sont les variables observables à l'aide de capteurs tandis que les entrées sont les variables directement modifiables à l'aide d'effecteurs.

Un système ne possédant qu'une entrée et qu'une sortie est dit *SISO* (Simple-Input and Simple-Output), il est qualifié de *MIMO* lorsqu'il en possède plusieurs (Multiple-Input and Multiple-Output).

La théorie des systèmes dynamiques exprime l'évolution d'un système au cours du temps comme le résultat d'une fonction prenant en argument son état (la valeur de ses entrées et sorties) ainsi que les actions appliquées sur ses entrées :

$$\frac{dx}{dt} = f(x, u)$$

où x est un vecteur comprenant les entrées et les sorties du système, u un vecteur comprenant les modifications effectuées sur les entrées et f représente le processus de transformation qui caractérise le système. La linéarité de cette fonction définit celle du système.

Un système est ainsi dit *linéaire* si son état évolue linéairement par rapport à son état précédent et aux actions sur ses entrées. Cela en fait un système relativement aisé à contrôler, en raison de l'absence de minimums locaux sur f.

La classe des systèmes *affines en contrôle* est un peu plus difficile à gérer. Ces systèmes évoluent linéairement par rapport aux contrôles appliqués et non-linéairement par rapport à leur état courant (Sontag 1998) :

$$\frac{dx}{dt} = f(x) + g(x)u$$

où f et g sont des fonctions quelconques, potentiellement non-linéaires. La composante linéaire de ces systèmes permet toutefois l'application de méthodes mathématiques de linéarisation facilitant par la suite le contrôle.

Enfin, les systèmes non-linéaires sont les systèmes dont l'évolution dépend de manière non-linéaire de son état comme des actions sur les entrées.

La non-linéarité fait partie des critères définissant les systèmes *complexes*. La complexité découle également de la difficulté, voire de l'impossibilité, d'identifier tous les mécanismes en jeu dans le système, et donc de prévoir son comportement. Certains des éléments et des relations du système, ou de son environnement, peuvent être incertains ou inaccessibles. Cela peut apparaître lorsqu'un très grand nombre d'éléments sont en jeu, que le graphe des relations n'est pas trivial (c'est-à-dire qu'il présente des cycles et que certains liens sont privilégiés), ou que certaines variables sont trop dynamiques ou physiquement non mesurables. En outre, ce type de systèmes présente fréquemment plusieurs niveaux : les éléments interagissant localement sont eux-mêmes des systèmes plus ou moins complexes, compliquant considérablement l'étude du système global.

Dans ce type de cas, les techniques classiques de contrôle telles que celles présentées en début de chapitre 2 ne peuvent s'appliquer efficacement, ce qui fait que le contrôle de systèmes complexes est un domaine de recherche actif.

1.1.2.2 La notion de contrôle

Le rôle d'un contrôleur est de transformer les souhaits de l'utilisateur en actions adéquates sur un système. On appelle *consigne* la valeur que les sorties doivent atteindre et *commande* les actions du contrôleur sur le système contrôlé, lui-même souvent désigné par le terme *procédé*. Lorsque le contrôleur ne bénéficie d'aucun retour sur les actions qu'il effectue, comme illustré par la figure 1.2, le contrôle est dit en *boucle ouverte*.

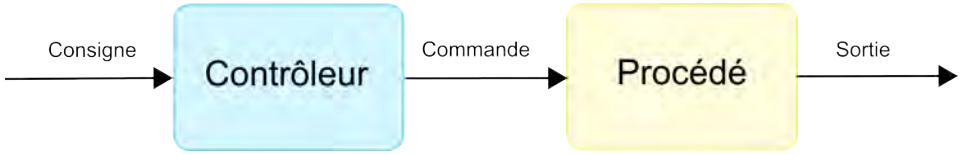

FIGURE 1.2 – Contrôle en boucle ouverte.

Le contrôle d'un système est donc la manipulation adéquate de ses entrées en regard des buts de l'utilisateur sur ses sorties. De cette définition générique découlent un certain nombre de sous-problèmes de "contrôle" à plusieurs niveaux. Un exemple du quotidien illustre bien cette situation : les véhicules motorisés. Si l'utilisateur d'une voiture désire accélérer, il va en donner la consigne par le biais de la pédale d'accélération. Plusieurs commandes doivent être entreprises sur le moteur pour respecter cette consigne. Ces commandes impliquent des actions de la part de différents effecteurs (valves, injecteurs, etc), qui doivent à leur tour être contrôlés afin de s'assurer que leur action (angle d'ouverture, durée d'injection, etc) est correcte et coordonnée avec les autres actions en cours.

Le contrôle peut comprendre la résolution de contraintes (maintenir des sorties au-dessus ou au-dessous d'un certain seuil) ainsi que des objectifs d'optimisation (minimiser ou maximiser des sorties). Lorsque la consigne est fixe (il s'agit de maintenir le procédé dans un état stable), le contrôle est appelé régulation. On parle de contrôle *robuste* lorsque celui-ci doit s'adapter à des changements dans le systèmes contrôlé, tel que des pannes ou du bruit sur les données.

La cybernétique, se focalisant sur les échanges d'information, a défini des concepts importants tels que ceux de *boîte noire* et de *rétroaction* (Wiener 1948).

Une boîte noire est un système dont l'intérieur est inaccessible. On ne peut connaître ni sa structure ni son fonctionnement interne. Il faut se reposer sur l'observation de ses sorties pour en déduire sa fonction. C'est un point de vue adopté par un certain nombre de systèmes de contrôle, par exemple ceux qui s'appliquent à des procédés trop complexes pour être modélisés.

La rétroaction (ou *feedback*, en anglais) est l'action que produit un phénomène en retour sur ses propres causes. Une rétroaction est dite positive si elle tend à amplifier le phénomène, et négative si elle tend au contraire à l'amortir. Dans le cadre du contrôle de système, la rétroaction se traduit par le fait que le contrôle d'un système est influencé par l'erreur entre la sortie du système et la consigne. Un exemple bien connu est celui du thermostat. Les contrôleurs basés sur le principe de rétroaction sont dits en *boucle fermée* (figure 1.3), ils constituent l'immense majorité des systèmes de contrôle actuels. Ainsi, le comportement du système contrôlé influence celui de son contrôleur et réciproquement. Les deux systèmes sont couplés.

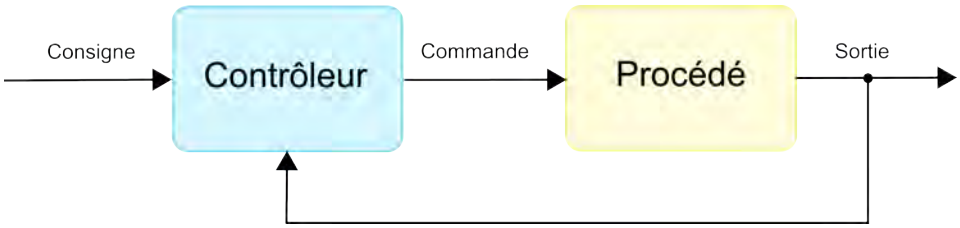

Figure 1.3 – Contrôle en boucle fermée.

La cybernétique apporte un éclairage important sur ce couplage entre contrôleur et procédé. La variété, notion analogue à celle de complexité, est le nombre d'états et de comportements différents que peut prendre un système donné. La loi de la variété requise stipule que la variété du système de contrôle doit être supérieure ou égale à celle du système contrôlé (Ashby 1956). Cette loi s'explique assez bien de manière intuitive. Le rôle d'un contrôleur est d'amener et de maintenir un système dans un état désiré. Il faut donc qu'à chaque état du système contrôlé corresponde un état du contrôleur en mesure de le gérer. Si cette condition n'est pas respectée, il peut se trouver des situations dans lesquelles le contrôleur n'est plus "maître" de la situation. Il est pris au dépourvu par un comportement inédit du procédé contrôlé, et il se produit alors une inversion du contrôle.

La section suivante s'intéresse au contrôle appliqué aux moteurs à combustion interne dans l'industrie automobile. Ce champ d'application a constitué le terrain d'expérimentation des travaux de cette thèse et a permis de nous confronter à des problèmes bien concrets.

1.2 Le contrôle de moteurs à combustion

Les moteurs à combustion sont des systèmes non-linéaires, dont les sorties sont fortement bruitées et sur lesquelles l'effet d'une action en entrée peut se manifester sur diverses échelles de temps. Contrôler un moteur permet d'en optimiser le comportement mais demeure une tâche complexe. Cette section en expose les principales caractéristiques.

1.2.1 L'unité de contrôle moteur

Les avancées technologiques des moteurs à combustion, aussi bien essences que diesels, ont permis d'en améliorer significativement les performances. Mais ces dernières, couplées au durcissement des normes anti-pollution, ont également provoqué la complexification de leur contrôle.

Le contrôleur d'un moteur est un calculateur embarqué, appelé unité de contrôle moteur (ou ECU, pour *Engine Control Unit*), sur lequel s'exécute un logiciel appliquant des stratégies de contrôle. Son rôle est de transformer la demande de couple issue de la pédale d'accélérateur en actions sur les effecteurs du moteur de manière à ce que le couple souhaité soit obtenu. Ce faisant, le contrôleur doit assurer le respect de nombreux critères hétérogènes et parfois contradictoires, tels que le respect de seuils de pollution et les besoins d'optimisation (maximisation de la puissance développée, minimisation de la consommation de carburant).

Le nombre et la nature des entrées et des sorties d'un moteur varie avec ses caractéristiques et ses fonctionnalités (par exemple le nombre de cylindres, la présence ou non d'une vanne EGR[1], etc). La figure 1.4 montre un exemple d'ECU couplé à un moteur essence avec les entrées et sorties de chacun des systèmes. De nombreux paramètres influencent le comportement d'un moteur, mais tous ne sont pas contrôlables par un ECU (par exemple la pression atmosphérique). De même, toutes les sorties d'un moteur ne sont pas nécessairement mesurables par l'ECU, selon l'instrumentation du moteur (en particulier la concentration des divers gaz ou la température et la pression en divers points nécessitent l'utilisation de sondes dont un moteur série n'est pas équipé). Le contrôleur doit donc gérer avec l'indisponibilité de certaines mesures. Le plus souvent celles-ci sont estimées à l'aide de modèles mathématiques des phénomènes physiques en jeu localement.

Le logiciel d'un ECU est généralement composé de deux couches. La couche haute transforme la consigne de couple en commandes de "haut niveau" (comme la durée des injections, le débit de carburant ou encore l'avance à l'allumage) que la couche basse traduit en actions sur le moteur (ouverture/fermeture de valves, etc). Ces couches sont elles-mêmes constituées de "sous-systèmes de contrôle" interdépendants, correspondant aux blocs fonctionnels propres

1. *Exhaust Gas Recirculation, il s'agit d'un circuit particulier permettant la réutilisation des gaz d'échappement comme comburant et visant à réduire l'émission d'oxydes d'azote.*

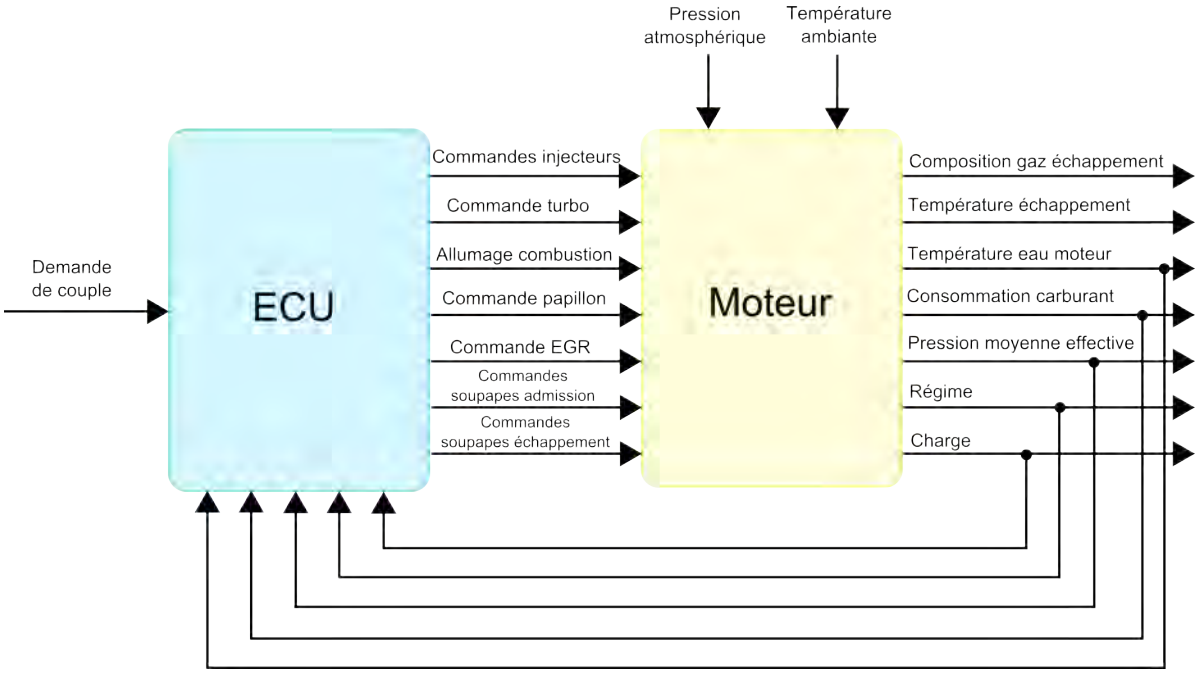

Figure 1.4 – Exemples d'entrées et de sorties d'un moteur essence et de son ECU.

à chaque moteur (débimétrie air, débimétrie essence, distribution, etc). Une troisième couche de contrôle, cette fois-ci électronique, s'assure que les actions calculées par la couche logicielle basse sont correctement effectuées. Bien entendu, cette thèse en informatique se concentre uniquement sur la partie logicielle de l'ECU.

1.2.2 La mise au point d'un ECU

Le développement du logiciel d'un ECU présente les difficultés classiques de l'optimisation multi-critère. En effet, le respect des réglementations sur l'émission de polluants ainsi que les diverses optimisations sont gérées durant la phase de mise au point. Les stratégies de contrôle sont élaborées manuellement à partir de l'expertise des ingénieurs. Elles définissent les calculs menant à la production des diverses consignes et commandes. Elle s'appuient notamment sur des tables, les cartographies, qui mettent le plus souvent en relation la valeur courante de la charge et du régime avec celle d'une variable de contrôle (par exemple l'avance à l'allumage). Il faut alors trouver les bonnes valeurs à affecter à ces cartographies de manière à ce que la réponse du moteur soit celle souhaitée. C'est la calibration.

Les valeurs de la charge et du régime définissent un point de fonctionnement moteur. Pour chaque point de fonctionnement, on cherche notamment à maximiser le couple, à minimiser la consommation et à rester en dessous des seuils de polluants (oxydes d'azote, particules, dioxyde de carbone) et de température (afin d'éviter la surchauffe). Ces critères sont souvent contradictoires, il faut donc trouver les réglages offrant un "bon" compromis, lui-même à définir par l'ingénieur. Par exemple, on peut obtenir un couple maximum très haut au prix

d'une plus grosse consommation. Le meilleur compromis dépend de l'utilisation future du moteur.

La calibration de l'ECU vise donc à optimiser le comportement du moteur. Elle demande de nombreux essais moteur, allant des différents bancs aux essais en roulage (et souvent de nombreux allers-retours entre ces étapes), afin de caractériser le moteur, c'est-à-dire d'identifier des points de fonctionnement intéressants, de connaître son comportement sur ces points, et d'alimenter les cartographies en conséquence. Le réglage définitif est obtenu en balayant manuellement un premier paramètre tout en fixant les autres. Sa valeur est fixée lorsque la sortie considérée (couple, consommation, etc) atteint sa "meilleure" valeur. Un second paramètre est ensuite balayé en suivant la même procédure, et ainsi de suite. Des retours sur un paramètre déjà arrêté sont éventuellement nécessaires si un optimum de la chaîne n'est pas capable de satisfaire certains critères importants.

Seule la demande de couple est une consigne prévue pour être dynamique durant le fonctionnement du contrôleur. Les autres consignes (les contraintes et les objectifs d'optimisation) sont statiques et prises en compte durant la phase de mise au point. Aussi, l'ECU doit être à nouveau calibré si les critères d'optimisation sont modifiés, ou lorsque le comportement du moteur évolue sous l'effet de l'usure et de l'encrassement de ses pièces mécaniques.

La nature complexe du comportement d'un moteur, le nombre important de paramètres (provenant tout autant du nombre d'effecteurs que des stratégies de contrôle en cascade rigides), et les critères variés et parfois divergents rendent la calibration très difficile. Pour un moteur nouveau, la mise au point dure plusieurs mois, parfois plus d'un an. C'est une procédure que les industriels cherchent donc à accélérer.

1.3 Objectifs de la thèse

Produire un ECU complet permettant le prototypage rapide de fonctions de contrôle moteur était justement le but du projet auquel les travaux de cette thèse ont pris part. Le projet Orianne impliquait en particulier la définition de stratégies de contrôles génériques et le développement d'un logiciel de calibration automatique de ces stratégies.

Concevoir et développer un programme informatique capable de calibrer automatiquement le calculateur était ainsi l'objectif premier de cette thèse. Dans ce contexte, calibrer signifie trouver la valeur adéquate des paramètres de l'ECU telle que les sorties du moteur sont conformes aux attentes. Cela revient en fait à effectuer le contrôle du système composé de l'union de l'ECU et du moteur. Les entrées de ce système sont les paramètres de calibration de l'ECU et les sorties sont celles que l'on observe sur le moteur (figure 1.5).

Toutefois, il n'est ici pas question de fournir à ce nouveau contrôleur des stratégies et des modèles préétablis. Ce serait décaler le problème initial sans le résoudre (on en revient à la loi de la variété requise, qui interdit de réduire la complexité). Le contrôleur à développer doit être capable d'apprendre lui-même le contrôle. Autrement dit, le système à développer doit trouver la valeur adéquate des entrées du système contrôlé, assurant le respect de critères imposés par l'utilisateur, et ce sans l'aide information préalable sur le système contrôlé. Les

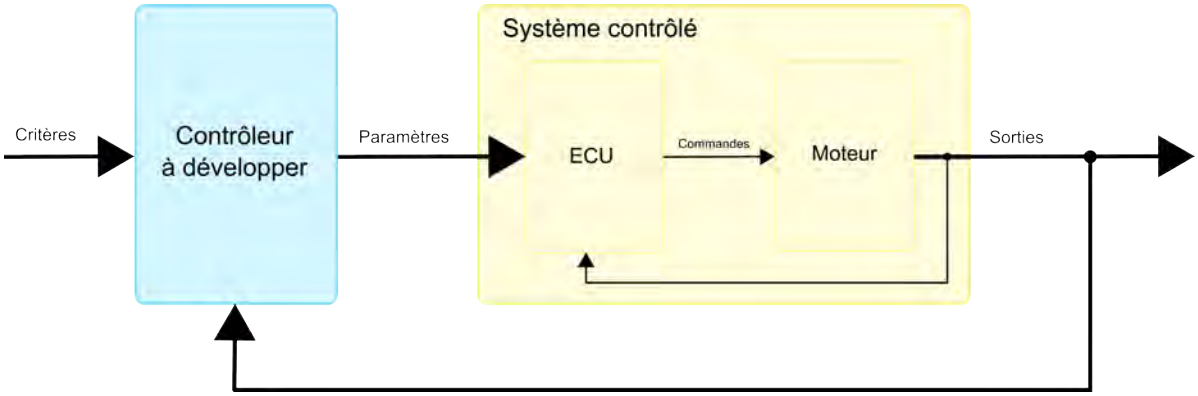

Figure 1.5 – Positionnement du système à développer.

seuls éléments renseignés a priori sont les critères à respecter sur les sorties, ceux-ci pouvant éventuellement changer dynamiquement. Le contrôleur doit ensuite apprendre comment les respecter en se basant uniquement sur l'observation en ligne des effets de ses actions.

C'est pourquoi l'objectif de cette thèse n'est pas tant la conception d'un système de calibration automatique mais bien celle d'un système de contrôle, avec la contrainte supplémentaire de ne requérir aucune information préalable sur le système contrôlé. Un tel système est ainsi générique, puisqu'il ne fait pas d'hypothèse sur le système contrôlé, et rapide à instancier à un procédé particulier, puisqu'il ne requiert ni modèle ni autre information préalable que les objectifs à atteindre.

Bien sur, plutôt que de trouver un paramétrage de l'ECU, un tel système peut en théorie remplacer le logiciel de ce dernier et contrôler directement un moteur. En pratique néanmoins, les fortes contraintes techniques de l'électronique embarquée ainsi que les besoins spécifiques liés au projet rendent irréaliste un tel objectif sur la durée d'une thèse.

1.4 Conclusion

Ce chapitre a introduit le domaine du contrôle de systèmes et présenté les objectifs de la thèse. Calibrer automatiquement revenant à contrôler et apprendre simultanément, ces objectifs consistent donc en la conception et l'implémentation d'un contrôleur générique, facile à instancier et capable d'apprentissage.

Le chapitre suivant explore les méthodes de contrôle et les techniques d'apprentissage actuelles afin d'en étudier la compatibilité avec les objectifs fixés et de préciser la définition de ces derniers.

CHAPITRE 2

Contrôle de systèmes complexes et apprentissage artificiel

L'objectif de ce chapitre est de donner une vue d'ensemble à la fois des techniques actuelles de contrôle de systèmes complexes et des méthodes d'apprentissage artificiel, qui semblent de prime abord pertinentes pour aborder la problématique évoquée au chapitre précédent.

Les techniques de contrôle se répartissent en trois grandes familles. Nous commencerons par les approches classiques : les contrôleurs PID, qui sont les plus répandus dans l'industrie, et les contrôleurs adaptatifs, plus efficaces sur les systèmes complexes. Les contrôleurs intelligents, faisant appel à des techniques d'intelligence artificielle (IA) pour améliorer leurs performances, seront introduits après avoir fait un tour d'horizon des méthodes d'apprentissage automatique dont ils usent. Ces approches de contrôle seront évaluées selon leur capacité à être appliquées à de nombreux domaines (généricité), leur facilité d'instanciation à un procédé particulier, leur capacité à suivre l'évolution dans le temps du système contrôlé (adaptativité) ainsi que leur faculté à apprendre le fonctionnement du procédé ou au contraire leur besoin de connaissances a priori (apprentissage). Enfin nous ferons un point sur les méthodes actuellement appliquées aux moteurs.

2.1 Les classiques du contrôle de systèmes

Le contrôle de systèmes étant un domaine étudié depuis de nombreuses décennies, les méthodes proposées sont très diverses et se sont largement ramifiées, raffinées et combinées au cours du temps. Celles présentées dans cette section, même si elles font toujours l'objet de travaux de recherche, sont bien implantées dans l'industrie et sont considérées comme classiques.

2.1.1 Contrôleurs PID

Les contrôleurs PID sont très largement répandus dans l'industrie. Ils tirent leur nom des trois termes qu'ils calculent à partir du signal de rétroaction afin de le transformer en correction à appliquer sur la variable contrôlée : Proportionnelle, Intégrale et Dérivée (Astrom et Hagglund 1995).

- Le terme Proportionnel, aussi appelé gain, permet de tenir compte de l'erreur actuelle entre la valeur observée du signal de rétroaction et la consigne.

  $$P = K_p e(t)$$

  où K_p est une constante, e l'erreur actuelle et t le temps. Augmenter l'importance de ce terme permet une convergence plus rapide vers la consigne mais dégrade rapidement la stabilité du système.
- Le terme Intégral sert à rendre le contrôle plus précis en prenant en compte l'erreur sur la durée.

  $$I = K_i \int_0^t e(\tau) d\tau$$

  où K_i est une constante, e l'erreur, t le temps, et τ la variable d'intégration. Augmenter l'importance de ce terme tend à éliminer l'erreur statique, mais ralentit fortement le temps d'établissement d'un régime stationnaire car cela provoque un dépassement et des oscillations.
- Enfin, le terme Dérivé permet de considérer le taux de variation de l'erreur et donc d'avoir une estimation de sa future valeur. Ainsi, le contrôleur peut par exemple diminuer l'amplitude de sa correction lorsque l'erreur diminue et inversement.

  $$D = K_d \frac{de(t)}{dt}$$

  où K_d est une constante, e l'erreur et t le temps. Augmenter l'importance de ce terme tend donc à diminuer le dépassement et les oscillations, mais fait perdre en précision. En effet, l'amplitude de la correction diminuant avec l'erreur, il perdure une différence entre la réponse et la consigne, que l'on appelle erreur statique.

Ces trois termes sont ensuite combinés selon une méthode propre à l'instance de PID considérée. La plus classique est la somme, la formule générale de la commande u(t) d'un PID est alors exprimée comme :

$$u(t) = K \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right) \tag{2.1}$$

où K, T_i, T_d sont des constantes appelées respectivement gain, temps d'intégration et temps de dérivation. Notons que selon les cas les fonctions I ou D peuvent être omises, on parle alors de contrôleurs P, PI ou PD.

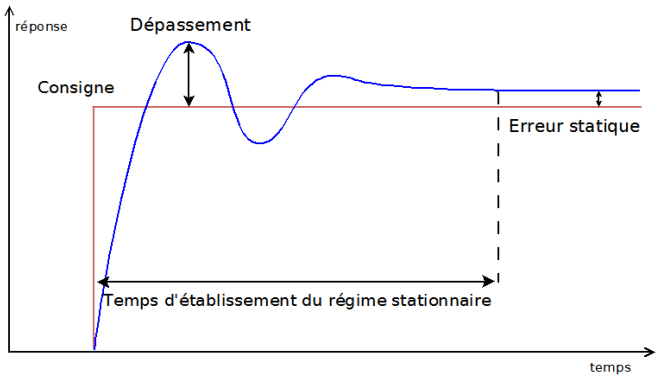

FIGURE 2.1 – Réponse d'un PID.

La figure 2.1 montre une réponse typique d'un PID (mal réglé) sur un procédé stable. Aussi, pour fonctionner correctement sur un système particulier, un contrôleur PID nécessite d'être paramétré avec justesse. Trouver les valeurs des constantes qui permettent un bon compromis entre rapidité et précision est un problème compliqué qui est parfois la raison principale de la désactivation d'une des fonctions. De nombreuses méthodes ont été développées pour faciliter le paramétrage des PID, la plus connue étant certainement la méthode de Ziegler-Nichols.

2.1.1.1 Méthode de Ziegler-Nichols

Ziegler et Nichols ont étudié l'influence des paramètres K, T_i, T_d de l'équation 2.1 afin de fournir aux ingénieurs une technique plus simple de paramétrage (ZIEGLER et NICHOLS 1942). La méthode consiste à désactiver les fonctions Intégrale et Dérivée du PID, par exemple en définissant $T_i = +\infty$ et $T_d = 0$. À consigne fixe, le gain K est ensuite mis à zéro puis progressivement augmenté, jusqu'à ce qu'il atteigne une valeur critique K_{crit} pour laquelle la réponse oscille de manière périodique autour d'une valeur stable. On note T la période de ces oscillations. Le tableau 2.1 donne les valeurs à appliquer aux paramètres selon le type de contrôleur (P, PI ou PID).

TABLE 2.1 – Ajustement d'un contrôleur PID selon la méthode de Ziegler-Nichols.

Contrôle	K	T_i	T_d
P	$K_{crit}/2$		
PI	$K_{crit}/2.2$	$T/1.2$	
PID	$K_{crit}/1.7$	$T/2$	$T/8$

Cette méthode donne des résultats globalement bons mais n'est pas toujours optimale. En outre, l'exploration des valeurs de K peut amener le procédé vers des états instables et donc rendre l'application de la méthode impossible à certains systèmes. De nombreuses variations ont été développées, telles que celles de VALÉRIO et SÁ DA COSTA 2006 ou de MUDI, DEY et LEE 2008, ou d'autres améliorant les résultats obtenus grâce à l'utilisation de la logique floue (VISIOLI 2001).

2.1.1.2 Limites des PID

Étant appliqués à de nombreux domaines, la généricité des PID n'est plus à démontrer. Cependant, elle est obtenue au prix d'un lourd travail de paramétrage demandant une connaissance approfondie aussi bien du fonctionnement du contrôleur que du procédé contrôlé. En outre, la fiabilité du matériel est importante puisque les PID réagissent mal aux imprécisions sur les mesures ainsi qu'à l'évolution au cours du temps du procédé contrôlé. Pour répondre à ce problème, des méthodes de paramétrage dynamique, permettant de mettre à jour les paramètres à la volée, ont été étudiées mais restent limitées à certaines classes de procédés (Chang, Hwang et Hsieh 2002) ce qui en limite fortement le caractère générique.

Utilisés seuls, les contrôleurs PID suffisent pour les systèmes dont la dynamique est inférieure au second ordre et où les exigences sur les performances du contrôle sont peu importantes. Sinon, il est nécessaire de leur adjoindre des mécanismes permettant d'optimiser leur comportement, parmi lesquels on note la logique floue (Carvajal, Chen et Ogmen 2000), la mise en cascade de PID (Lee, Park et Lee 1998), l'implémentation sous forme de réseau de neurones artificiels (Shu et Pi 2000), ou encore le principe du *feed-forward*, c'est-à-dire l'utilisation de connaissances extérieures relatives au procédé contrôlé (Mizumoto et al. 2010). Ces raffinements font pour la plupart partie du "contrôle intelligent" qui sera abordé plus tard.

Enfin, une limite importante à l'utilisation des PID est que l'algorithme de base ne prévoit qu'un seul signal de rétroaction et ne contrôle qu'une seule variable. Des méthodes de composition de plusieurs PID permettant de gérer des systèmes MIMO existent (Ayadi et Benhadj 2005), mais leur instanciation n'en est que plus difficile.

2.1.1.3 Bilan des PID

En conclusion, l'algorithme de base des PID est générique mais limité aux systèmes SISO, de faible dynamique et stables dans le temps. Ses raffinements lui permettent de s'étendre à d'autres classes de problèmes mais souffrent en contrepartie de la nécessité d'un lourd travail pour être appliqués à un procédé particulier. En outre cette instanciation demande de la part de l'utilisateur de bonnes connaissances sur le système contrôlé. Ces remarques sont récapitulées par le tableau 2.2.

Table 2.2 – Bilan des PID.

Critère	PID
Généricité	+
Instanciation	- -
Adaptativité	- -
Apprentissage	Aucun (la connaissance du procédé est implicitement contenue dans le paramétrage)

Les PID constituent souvent la couche basse de l'implémentation de systèmes de contrôle plus évolués. Ces systèmes, gérant plusieurs variables et une dynamique plus complexe, produisent des consignes envoyées aux PID qui contrôlent chacun une seule variable spécifique.

2.1.2 Contrôle adaptatif

Les caractéristiques de certains procédés évoluent au cours du temps, leur contrôle nécessite alors d'être réajusté. Un exemple courant est la température d'un moteur à combustion, augmentant avec le temps lorsque le moteur est en fonctionnement et influant grandement sur son comportement. Les méthodes s'attaquant à ce problème sont regroupées sous le terme de contrôle adaptatif. Nous abordons ici les principales approches à savoir l'utilisation d'un modèle de référence, l'identification de modèle, le contrôle dual et le contrôle par apprentissage itératif.

2.1.2.1 Contrôle avec modèle de référence

Les systèmes de contrôle adaptatifs à modèle de référence (*Model Reference Adaptive Controller*, ou MRAC, en anglais) comprennent trois briques : un modèle mathématique du procédé contrôlé, un mécanisme d'ajustement de paramètres et le contrôleur à proprement parler (Whitaker, Yamron et Kezer 1958).

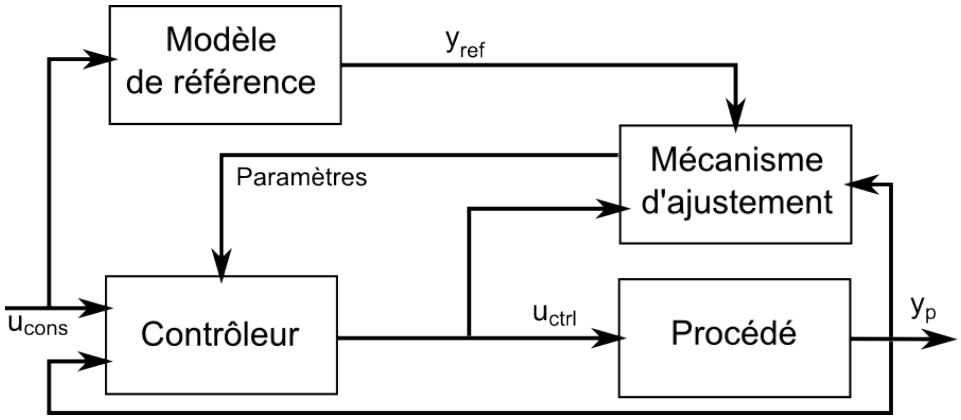

Figure 2.2 – Schéma d'un système MRAC.

Le schéma général d'un système MRAC est donné par la figure 2.2. L'idée de base est d'ajuster les paramètres du contrôleur (et donc modifier les commandes u_{ctrl} qu'il produit) afin que la sortie du procédé y_p se comporte comme celle de son modèle "idéal" y_{ref}. Autrement dit, l'ajustement des paramètres du contrôleur compense l'évolution du procédé. La consigne u_{cons} n'a donc pas besoin d'être modifiée. C'est là une utilisation assez inhabituelle d'un modèle puisqu'on ne cherche pas à l'ajuster pour qu'il reste fidèle au procédé, mais on ajuste le contrôle du procédé pour qu'il reste fidèle au modèle.

Le mécanisme d'ajustement est généralement basé sur un processus de type descente de gradient comme la *MIT rule* (du nom du célèbre institut), minimisant l'erreur entre y_{ref} et y_p

(Mareels et al. 1986). Il doit évidemment être choisi en adéquation avec le type de contrôleur utilisé, qui peut par exemple être un PID sous réserve de quelques ajustements (Braziunas 1992).

2.1.2.2 Contrôle avec identification de système

Plutôt que de se servir d'un modèle comme d'une référence à suivre, les systèmes de contrôle à identification de modèle (*Model Identification Adaptive Controllers*, ou MIAC, en anglais) cherchent à ajuster la représentation qu'ils ont du système contrôlé pour pouvoir paramétrer dynamiquement le contrôleur afin qu'il demeure adapté au procédé (Sastry et Bodson 1994). Les briques qui les composent sont semblables à celles des systèmes MRAC mais ne jouent pas exactement le même rôle (figure 2.3).

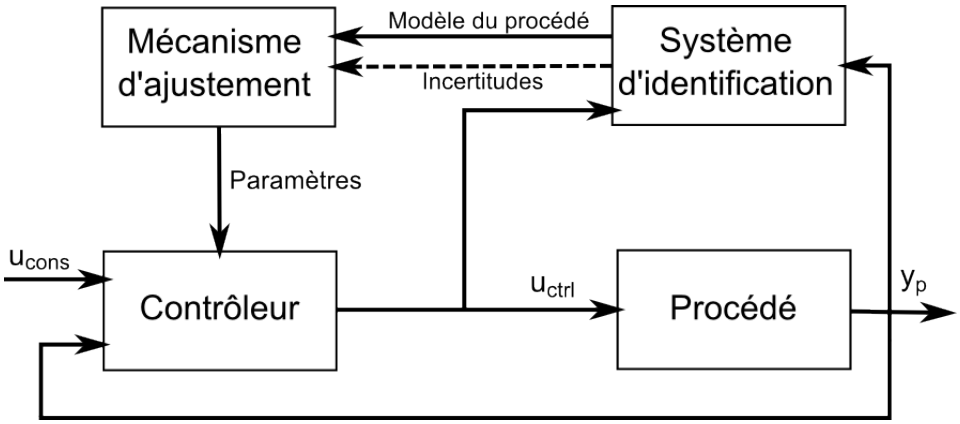

Figure 2.3 – Schéma d'un système MIAC.

Le mécanisme d'ajustement se base sur la connaissance préalable de la relation entre les paramètres du modèle identifié et ceux du contrôleur. Il ajuste dynamiquement les paramètres du contrôleur en fonction des informations que lui donne le système d'identification.

Branché sur les entrées et les sorties du procédé, le système d'identification a pour rôle d'en édifier et d'en maintenir un modèle. Ce modèle peut-être construit avec ou sans apport de connaissances préalables (on parle alors respectivement d'identification de boîte grise ou de boîte noire). L'identification de systèmes est une discipline à part entière qui, bien que fortement liée au domaine du contrôle, fait l'objet de travaux qui lui sont propres (Soderstrom et Stoica 1988). On y retrouve des méthodes de régression linéaire comme les moindres carrés (Chen, Billings et Luo 1989), de filtrage (Van Der Merwe et Wan 2001) ou encore des méthodes dites non-paramétriques basées sur l'analyse des entrées et sorties du procédé (analyse temporelle, analyse fréquentielle, etc) (Pintelon et Schoukens 2004), ainsi que des techniques d'intelligence artificielle comme la logique floue ou les réseaux de neurones (Xue et al. 2012). Du choix du système d'identification dépendra le mécanisme d'ajustement de paramètres, et donc indirectement, la loi de contrôle implémentée.

Notons enfin que MIAC comme MRAC constituent avant tout une architecture de contrôle et non une loi de contrôle décrivant exactement quelles actions sont à entreprendre sur le procédé comme peut l'être un PID.

2.1.2.3 Commande prédictive

Une variante largement répandue de système de contrôle avec modèle est la commande prédictive (*Model Predictive Command*, MPC, en anglais). Ici, la fonction du modèle est de prévoir les futures réactions possibles du système contrôlé. En explorant, grâce au modèle, l'espace d'états du procédé sur un horizon de temps fini, un algorithme d'optimisation calcule les meilleurs prochains contrôles à appliquer. L'algorithme général est représenté par la figure 2.4 (Nikolaou 2001).

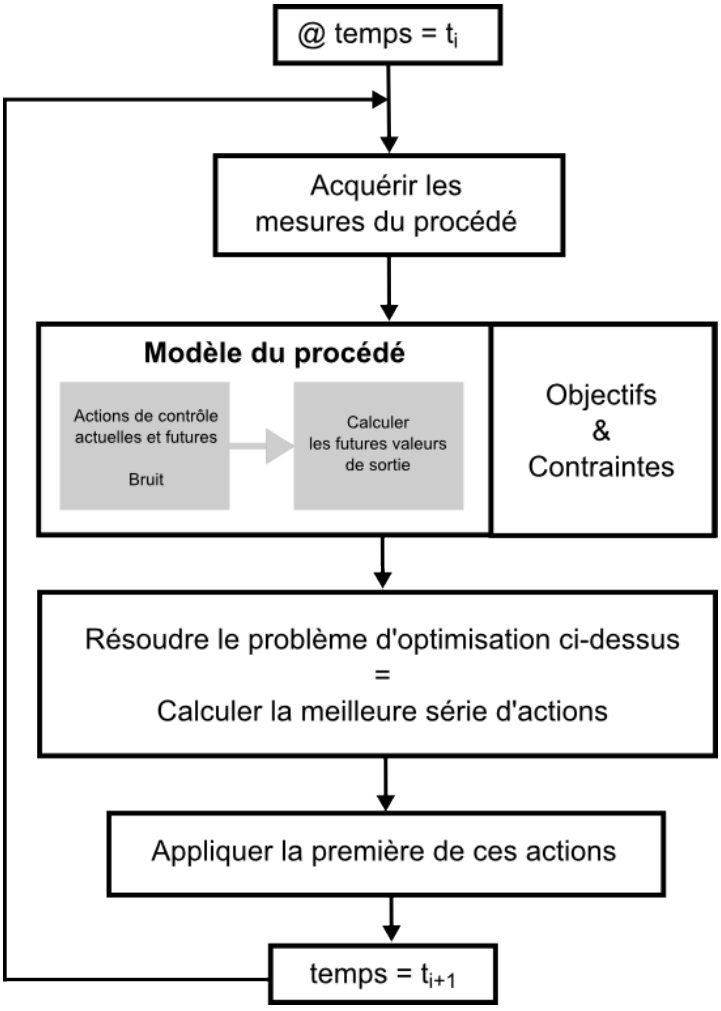

Figure 2.4 – Algorithme général de la commande prédictive.

Cette méthode permet de gérer plusieurs entrées et sorties, et de prendre en compte des contraintes de manière explicite tout en maintenant des objectifs à plus ou moins long terme.

Ses performances dépendent fortement de l'algorithme d'optimisation choisi. Celui-ci peut par exemple utiliser un modèle inverse du procédé pour calculer les entrées à appliquer à partir des sorties souhaitées.

La commande prédictive fonctionne de manière satisfaisante lorsque le modèle utilisé est linéaire, mais cela signifie souvent se limiter à une plage de fonctionnement réduite du procédé sur laquelle son comportement est correctement approximé par un modèle linéaire. Contrôler le système sur l'ensemble de ses plages de fonctionnement demande de passer à un modèle non-linéaire. On parle, dans ce cas, de *Non-linear Model Predictive Commade* (NMPC). En faisant apparaître des minimums locaux dans le problème d'optimisation, la non-linéarité du modèle complique considérablement son exploitation. Les principales solutions à cette difficulté consistent à reformuler le problème d'optimisation en linéarisant le signal de rétroaction reçu, souvent à l'aide d'un changement de variable (Isidori 1999). Cependant cette technique n'est utilisable que sur un sous-ensemble des systèmes non-linéaires : les systèmes "affines en contrôle", c'est-à-dire dont l'évolution de l'état dépend linéairement des contrôles appliqués et non-linéairement de l'état précédent. En outre, elle complique significativement la prise en compte de contraintes (Deng, Becerra et Stobart 2009).

Enfin, lorsque le système à contrôler est large (c'est-à-dire lorsqu'il a de nombreuses entrées et sorties), il devient souvent impossible de le gérer dans son ensemble de manière centralisée. Dans une approche récente, la commande prédictive distribuée (*Distributed MPC*), chaque sous-partie du procédé est gérée localement par un contrôleur MPC. Chacun de ces contrôleurs échange des informations avec ses voisins, par exemple la trajectoire prévue par son modèle, et ils parviennent ensemble à garantir une certaine stabilité du système contrôlé (Müller, Reble et Allgöwer 2011). Le découpage en sous-parties et la définition des contrôleurs locaux demeure un problème ouvert dans le cas général, mais l'idée de distribuer le contrôle est pertinente et nous aurons l'occasion de l'aborder à nouveau dans ce document.

2.1.2.4 Contrôle dual

Le contrôle dual (*Dual Control Theory*) propose des bases pour le contrôle d'un système inconnu au départ et repose sur l'utilisation de deux types d'action de contrôle (Feldbaum 1961) :

- Les actions de contrôle effectives, ayant pour but d'amener le procédé dans l'état désiré, et basées sur les connaissances actuelles du contrôleur.
- Les actions "sondes", dont les conséquences sont analysées par le contrôleur pour affiner ses connaissances sur le système contrôlé.

Ici, et contrairement aux MIAC, le système de contrôle ne se base pas sur l'erreur entre les sorties d'un modèle et les observations passives du procédé pour acquérir de nouvelles informations sur le système contrôlé. Il entreprend directement des actions pour en extraire de l'information. La structure du système contrôlé est supposée connue, c'est-à-dire que le contrôleur dispose d'une fonction f (un modèle) représentant le procédé et calculant la sortie y à l'instant k+1 à partir du contrôle u effectué à l'instant k, de l'historique γ de toutes les actions et observations jusqu'à l'instant k , d'un vecteur de paramètres inconnus θ et d'un

processus stochastique ζ connu représentant l'évolution (des paramètres) du procédé.

$$y(k+1) = f(u(k), \gamma_k, \theta(k), \zeta(k))$$

Il reste alors au contrôleur à estimer correctement les paramètres de ce modèle grâce à des actions d'exploration.

Ce type de contrôle présente l'avantage de parvenir à converger rapidement vers l'état souhaité et convient donc bien aux cas où les paramètres du procédé évoluent trop rapidement pour un MIAC ou bien lorsque l'horizon de temps pour aboutir à un contrôle satisfaisant est court (Wittenmark 2002). En revanche, appliquer des actions exploratrices pour en apprendre les conséquences présente le risque de dégrader la qualité du contrôle, voire d'emmener le procédé dans un état à partir duquel il ne peut plus atteindre l'état souhaité. Or, plus le contrôleur applique des actions "sondes", meilleure est son estimation des paramètres du système et donc meilleur devient son contrôle sur le long terme. Il y a ainsi un équilibre à trouver entre les deux types d'action, correspondant à un compromis entre un contrôle correct à court terme et un contrôle plus fin à long terme. Ceci constitue une limite à l'applicabilité de cette technique car trouver cet équilibre revient à faire en sorte que le contrôleur satisfasse une équation particulière appelée *équation de Bellman* (2.2).

$$V(\zeta(k), k) = \min_{u(k-1)} E\{(y(k) - y_r(k))^2 + V(\zeta(k+1), k+1) | \gamma_{k-1}\} \tag{2.2}$$

$V(\zeta(k), k)$ peut être interprétée comme la perte minimum attendue à partir de l'instant k et pour la suite du contrôle entre la sortie observée $y(k)$ et la consigne $y_r(k)$, étant donné les observations acquises jusqu'à maintenant γ_{k-1}. E dénote l'espérance mathématique prise sur la distribution de ζ. Résoudre cette équation signifie trouver la meilleure action de contrôle $u(k-1)$, celle qui assurera une perte minimum pour la suite du contrôle. Cette action ayant une influence immédiate sur la sortie du procédé et différée sur l'estimation future des paramètres, la solution de l'équation est un choix entre une perturbation (une action "sonde") qui minimisera le terme $V(\zeta(k+1), k+1)$ (c'est-à-dire la prévision moyenne des prochaines pertes) ou une action visant à améliorer la perte immédiatement (c'est-à-dire minimiser le terme $(y(k) - y_r(k))^2$). Ceci s'avère très difficile dans la plupart des cas concrets, notamment en raison de la grande dimension de ζ.

Malgré tout, la recherche d'une corrélation entre variations sur les entrées et conséquences observables sur les sorties après l'application d'une action est une idée très intéressante qui mérite d'être conservée. Nous la retrouverons dans les chapitres suivants.

2.1.2.5 Contrôle par apprentissage itératif

Introduit en anglais pour la première fois par Arimoto, Kawamura et Miyazaki 1984, le contrôle par apprentissage itératif (*Iterative Learning Control*, ILC) a d'abord été motivé par la maîtrise de bras robotisés industriels, répétant indéfiniment la même tâche. Chaque nouvelle passe améliore le contrôle en s'appuyant sur la précédente.

$$u_{k+1}(t) = u_k(t) + [Ke_k](t)$$

Le nouveau contrôle appliqué u_{k+1} est donc fonction du précédent u_k, de la dernière erreur e_k entre la sortie et la consigne, et d'un "opérateur de gain" K propre à l'implémentation. Cet opérateur fait le plus souvent appel à un modèle du système contrôlé, représenté sous forme matricielle (Owens et Daley 2008).

Cette méthode est réservée aux procédés présentant un comportement périodique. En cela elle est comparable au contrôle répétitif (*Repetitive Control*, RC) et au contrôle *run-to-run* (R2R) (Wang, Gao et Doyle 2009).

2.1.2.6 Bilan du contrôle adaptatif

L'utilisation de modèles donne aux méthodes de contrôle adaptatif l'avantage de gérer naturellement plusieurs entrées et sorties. En outre, une fois couplés à des algorithmes d'optimisation (éventuellement dynamiques), ces modèles permettent de mettre à jour les paramètres d'un contrôleur (MRAC et MIAC), ou bien interviennent directement dans le calcul du meilleur contrôle à appliquer (MPC, Dual Control Theory, ILC). Ainsi, le contrôleur est capable de suivre l'évolution du procédé au cours du temps et d'adapter ses actions en conséquence, ce qui lui confère une certaine robustesse.

Mis à part l'apprentissage itératif qui n'est utilisable qu'avec un système répétitif, les approches de contrôle adaptatif sont suffisamment génériques pour être appliquées dans de nombreux domaines. On retrouve, par exemple, l'architecture MIAC dans des domaines aussi variés que le contrôle de l'énergie dans un bâtiment (Pargfrieder et Jörgl 2002) ou celui d'un robot en interaction avec un humain (Casals 2013). Cependant cette généricité est à modérer selon la disponibilité d'un modèle adéquat du système à contrôler.

En effet, si la modélisation joue un rôle clé dans les contrôleurs adaptatifs, elle est aussi la cause de leur principale limite : leur difficulté à être instanciés. Dans le cas d'un système à contrôler totalement inconnu au départ, établir un modèle peut demander des années d'études ainsi que de lourds moyens technologiques. Une fois le modèle obtenu, il faut le paramétrer pour qu'il corresponde parfaitement à l'instance considérée du procédé modélisé. Cette étape s'appelle la calibration. Dans un système de contrôle, cette calibration peut non seulement concerner le modèle utilisé, mais aussi les différents mécanismes d'ajustement et d'optimisation. Cette tâche est d'autant plus difficile lorsque le procédé et son modèle ne sont pas linéaires. Néanmoins, cette limite ne remet pas en cause les principes fondamentaux du contrôle adaptatif. Le tableau 2.3 synthétise les paragraphes précédents.

Table 2.3 – Bilan du contrôle adaptatif.

Critère	Contrôle adaptatif
Généricité	+
Instanciation	- -
Adaptativité	+
Apprentissage	Limité (ajustement de paramètres d'une structure fixe)

2.1.3 Bilan des approches classiques de contrôle

Nous remarquons que leur difficile instanciation constitue la limite commune des approches présentées jusqu'ici. Pour contourner les difficultés de la modélisation et de la calibration (qui sont les tâches principales à réaliser pour instantier un système de contrôle à son procédé) les travaux les plus récents incorporent au contrôle des techniques issues de l'intelligence artificielle. L'objectif est souvent de faire apprendre le modèle, ses paramètres ou encore sa propre calibration au contrôleur de manière automatique et avec le minimum de connaissances a priori. Dans d'autres cas, il peut aussi s'agir d'apprendre directement une loi de contrôle.

Pour faciliter la présentation de cette famille de contrôleurs, dits intelligents, et parce que l'apprentissage est une partie importante du système présenté dans le chapitre 4, il paraît nécessaire d'en introduire d'abord les principales techniques.

2.2 L'apprentissage artificiel

On parle d'apprentissage artificiel (ou apprentissage automatique) lorsqu'un programme a la capacité d'améliorer ses performances à partir de données acquises en cours de fonctionnement, c'est-à-dire à partir de son expérience (Mitchell 2006). Ce type de système est utilisé pour résoudre des tâches trop complexes pour les algorithmes classiques. L'apprentissage étant un sujet central depuis les débuts de l'intelligence artificielle, il en existe de nombreuses approches. Elles sont généralement regroupées en trois familles qui se différencient par le type d'information dont dispose le système pour apprendre et le protocole avec lequel il interagit avec son environnement : l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage par renforcement (Cornuéjols et Miclet 2010).

2.2.1 Apprentissage supervisé

L'apprentissage supervisé désigne les techniques se basant sur un *oracle* pour guider le système apprenant. Lors d'une première étape, l'oracle fournit des exemples étiquetés, c'est-à-dire un ensemble de m couples comprenant une donnée x et la sortie attendue pour cette donnée u. On appelle cet ensemble l'*échantillon d'apprentissage* et on le note S.

$$S = \{x_i, u_i\}_{1 \leq i \leq m} = \{x_i, f(x_i)\}_{1 \leq i \leq m} \tag{2.3}$$

La fonction f, parfois appelée *fonction cible*, n'est connue que de l'oracle et est l'objet de l'apprentissage du système. En exploitant cet échantillon, l'apprenant doit trouver (ou s'approcher de) la bonne sortie $u_n = f(x_n)$ correspondant à une donnée d'entrée x_n qui n'appartient pas nécessairement à S. On appelle la fonction ainsi estimée par le programme apprenant une *hypothèse*.

Un algorithme d'apprentissage supervisé cherche dans l'espace des hypothèses possibles celle qui est la plus adéquate en regard de l'échantillon d'apprentissage. Cependant, il existe une infinité de fonctions qui passent par un ensemble de points donnés. Et aucune de ces

fonctions n'est *a priori* meilleure qu'une autre pour décrire l'ensemble des points de S et interpoler les résultats attendus. Comment choisir celle qu'il faut retenir est un des problèmes fondamentaux de l'apprentissage artificiel. Il faut s'assurer que l'apprenant dispose d'un moyen pour le contraindre à converger vers une hypothèse qui conviendra aux données sur lesquelles il sera appliqué après apprentissage. Il s'agit par exemple de la marge maximale dans le cas des machines à vecteurs de support. De manière plus générale le principe du rasoir d'Occam, qui privilégie toujours la solution la plus simple, est une réponse possible. En fait, il s'agit de faire un compromis entre les hypothèses complexes qui correspondent mieux aux données et celles plus simples qui sont meilleures pour généraliser (Russell et Norvig 2010). Si l'hypothèse trouvée est trop complexe, le système est en sur-apprentissage : il ne fait plus d'erreur sur les données d'apprentissage, mais il en commet de plus en plus sur les exemples nouveaux à identifier, à l'image d'un étudiant qui apprend par cœur son cours sans arriver à généraliser. À l'inverse, une hypothèse trop simple ne rend pas suffisamment bien compte de la réalité des données. C'est ce qu'il arrive par exemple si l'on utilise une méthode de séparation linéaire sur un échantillon non linéaire.

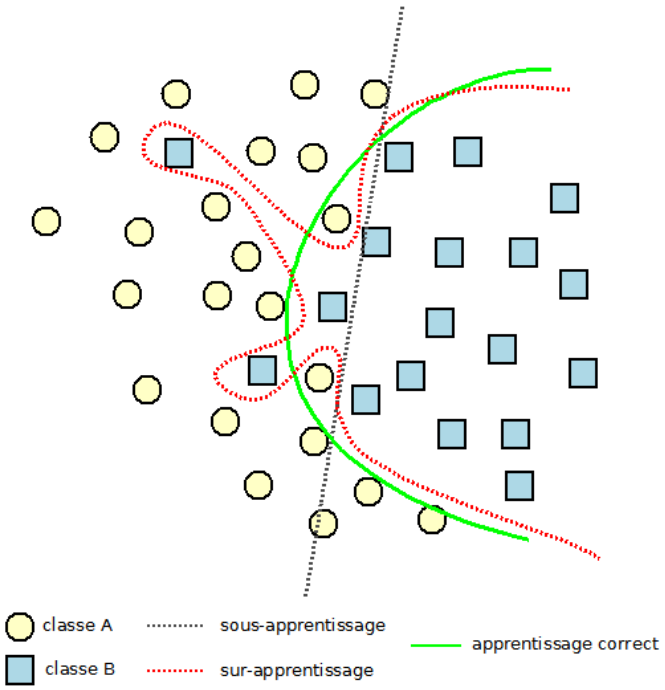

Figure 2.5 – Différents apprentissages de la séparation de 2 classes en 2 dimensions, sur un échantillon bruité.

La figure 2.5 montre un exemple de sur-apprentissage, de sous-apprentissage et d'apprentissage acceptable pour la séparation de deux classes en deux dimensions sur un échantillon de données bruitées. La courbe verte représente la séparation attendue. La courbe rouge est le résultat d'un sur-apprentissage : les données (y compris le bruit) sont parfaitement séparées, mais une nouvelle donnée à trier a des chances d'être mal classée (il n'y a pas

de généralisation). Enfin, la droite grise est le résultat d'un sous-apprentissage, l'hypothèse retenue est trop simple pour correspondre aux données.

L'apprentissage supervisé est classiquement utilisé pour les problèmes de classification ou encore la reconnaissance de formes. Sept des principales approches sont présentées ci-après.

2.2.1.1 Méthode des k plus proches voisins

Souvent abrégée en kPPV en français (ou kNN en anglais, pour k-Nearest Neighbors) la méthode des k plus proches voisins est simple mais efficace dans beaucoup de cas. Le principe est d'assigner à la donnée d'entrée la classe majoritaire parmi ses plus proches voisins dans l'échantillon d'apprentissage.

Il existe de nombreuses variantes de cette méthode, selon la fonction de distance utilisée ou encore selon la pondération des voisins entre eux. Sa principale limite est d'être coûteuse, notamment à cause de la recherche de voisins dans un échantillon potentiellement grand. Elle n'est donc pas adéquate dans les cas où un apprentissage dynamique est nécessaire. Une manière de réduire les coûts de calcul est de construire un modèle de l'échantillon sur lequel baser l'apprentissage (Guo et al. 2003) mais cela n'est pas toujours suffisant. En outre, la distribution des classes dans l'échantillon d'apprentissage peut biaiser le résultat final. En effet, si une classe est significativement plus présente que les autres, les chances sont grandes pour qu'elle soit majoritaire parmi les k plus proches voisins de la donnée testée (Coomans et Massart 1982). Ce biais peut être diminué par le choix d'une pondération adéquate (par exemple inversement proportionnelle à la distance). Enfin, le choix de k est également un problème récurrent. S'il est trop petit le bruit sur l'échantillon d'apprentissage impacte fortement le résultat, mais s'il est trop grand, les limites entre les classes sont moins bien définies.

2.2.1.2 Inférence d'arbres de décision

Les arbres de décision cherchent à classer un objet à l'aide d'une succession de tests sur ses attributs. Ces tests sont organisés hiérarchiquement, de manière à ce que la réponse à un test indique quel est le prochain à effectuer, et ainsi de suite jusqu'à ce que le dernier pointe sur la réponse finale. On aboutit ainsi à un arbre dont les nœuds sont des tests et les feuilles des classes (plusieurs feuilles peuvent correspondre à une même classe). Dans l'exemple de la figure 2.6, les objets sont définis par les attributs *sol* et *equipement* et sont à ranger parmi les classes *terrain de basket*, *terrain de rugby* et *terrain de football*. Beaucoup de connaissances du domaine sont nécessaires à la création de cet arbre, comme le fait que le rugby ne se joue pas sur du bitume.

Outre sa forme graphique, un arbre de décision peut être représenté par un ensemble de formules logiques. En effet, une branche allant de la racine jusqu'à une feuille est une conjonction de propriétés sur l'objet suffisante pour l'associer à une certaine classe. Par exemple, dans la figure 2.6, la branche menant à la feuille étiquetée *terrain de rugby* peut être décrite par la formule $(sol = gazon) \wedge \neg(equipement = cage)$. De même, une classe est représentable par la disjonction de toutes les branches menant à une feuille de même valeur.

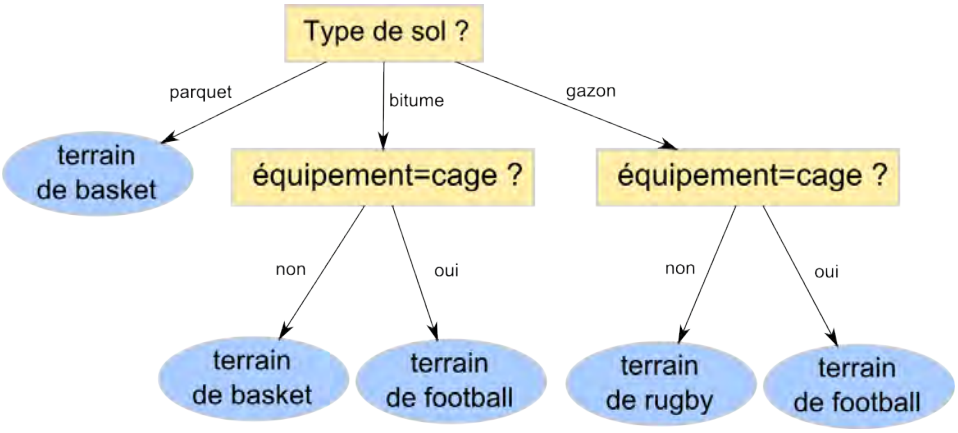

Figure 2.6 – Exemple d'arbre de décision.

Toujours dans le même exemple, la classe *terrain de basket* et ainsi représentée par la formule $(sol = parquet) \vee ((sol = bitume) \wedge \neg(equipement = cage))$.

En apprentissage supervisé, il s'agit de construire automatiquement un arbre de décision à partir de l'échantillon d'apprentissage. Une manière classique d'y parvenir est de récursivement partitionner l'échantillon selon la valeur de chaque attribut, créant ainsi un noeud pour chaque partition. Le processus se poursuit jusqu'à obtenir une partition ne contenant que des données associées à la même valeur de sortie, c'est-à-dire une feuille. Cette approche est connue sous le nom d'*induction descendante d'arbres de décision*. Les algorithmes fondateurs basés sur cette idée sont CART (Breiman et al. 1984) et ID3 (Quinlan 1986). L'espace des arbres valides pour un échantillon donné étant très grand, il faut l'explorer intelligemment et tenter d'en extraire un arbre efficace, c'est-à-dire suffisamment précis mais ne contenant pas trop de feuilles. L'ordre de sélection des attributs pour le partitionnement impacte fortement le résultat, aussi est il nécessaire d'avoir une métrique pour trier et sélectionner l'attribut le plus adéquat. Citons par exemple l'*entropie croisée* qui mesure le niveau de corrélation entre un attribut et la répartition des classes (Cover 1991).

L'avantage des arbres de décision est qu'ils sont souvent concis et compréhensibles. En outre, contrairement à la méthode des kPPV, la décision est peu coûteuse à prendre une fois l'arbre obtenu. Cependant leur utilisation impose une certaine structure de données compatible qui, selon le problème, peut être difficile à obtenir ou trop coûteuse à exploiter. Les graphes de décision sont une extension des arbres qui permet de limiter la redondance de certains nœuds (Oliver 1993). Les raffinements les plus récents se concentrent sur la prise en compte du coût de l'erreur de décision (Lomax et Vadera 2013).

En revanche, dans la pratique se pose le problème du sur-apprentissage. En laissant un algorithme basique d'induction se dérouler jusqu'au bout, l'arbre généré est trop précis : il est trop grand et chaque feuille est "pure" (c'est-à-dire qu'elle correspond à une même classe). Le bruit sur l'échantillon d'apprentissage est dans ce cas très gênant. C'est pourquoi il est nécessaire d'élaguer l'arbre. Cela peut être fait en jouant sur le critère d'arrêt de l'algorithme de construction (pré-élagage) ou bien en cherchant à remonter progressivement les feuilles

vers la racine tout en surveillant un critère d'erreur (post-élagage) (Patel et Upadhyay 2012). Ce problème demeure un sujet de recherche ouvert et constitue un obstacle à l'utilisation facile de cette méthode. Un autre frein majeur à son application sur des cas réels est sa faible robustesse face à des données manquantes.

2.2.1.3 Machines à vecteurs de support

Issues des travaux théoriques de Vapnik 1995, les machines à vecteurs de support sont également connues sous le nom de séparateurs à vaste marge (*Support Vector Machines* en anglais, SVM). Elles reposent principalement sur deux notions qui leurs sont préexistantes : les fonctions noyaux et la notion de marge maximale. Une fonction noyau est une densité de probabilité symétrique par rapport à l'axe des ordonnées, comme la loi de Gauss par exemple. En apprentissage supervisé, le problème est souvent de trouver la séparation entre des échantillons de classes différentes. On appelle *vecteurs de support* les échantillons les plus proches de la frontière de séparation. La marge désigne la distance qui les sépare de cette frontière et la théorie indique qu'il est préférable qu'elle soit maximale.

Dans le cas d'une séparation linéaire de deux classes, il s'agit de trouver l'hyperplan qui sépare les classes tout en maximisant la marge. L'équation d'un hyperplan étant :

$$h(x) = w.x + w_0$$

où w est le vecteur normal de l'hyperplan et w_0 une constante représentant son origine, un point (x_i, y) est bien classé si et seulement si

$$y.h(x_i) > 0$$

Une première expression du problème, dite primale, est alors la suivante :

$$\begin{cases} min(\frac{1}{2}\|w\|^2) \\ \forall i, u_i(w.x_i + w_0) \geq 1 \end{cases}$$

où w (le vecteur normal de l'hyperplan) et w_0 sont les paramètres à trouver et x_i et u_i les données de l'échantillon d'apprentissage. Tel quel le problème est difficile (voire impossible) à résoudre lorsque la dimension des données d'entrée est grande. Il est préférable de l'exprimer sous sa forme duale, qui ne dépend plus de la dimension des données mais de la taille de l'échantillon d'apprentissage et qui est :

trouver les multiplicateurs de Lagrange α tels que :

$$\begin{cases} \max\limits_{\alpha} \{\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j u_i u_j (x_i.x_j)\} \\ \alpha_i \geq 0, i = 1, ..., m \\ \sum_{i=1}^{m} \alpha_i u_i = 0 \end{cases} \tag{2.4}$$

où m est la taille de l'échantillon d'apprentissage. Le théorème de Kuhn-Tucker démontre que la solution au problème dual et celle au problème primal sont les mêmes (Kuhn et Tucker

1951). L'hyperplan solution est alors donné par :

$$h(x) = (w^*.x) + w_0^* = \sum_{i=1}^{m} \alpha_i^* u_i.(x.x_i) + w_0^*$$

où les α_i^* sont solutions de l'équation 2.4 et où w_0^* peut être calculé à partir d'un *vecteur de support*. Les *vecteurs de supports* sont les seuls à avoir un multiplicateur de Lagrange non nul, ils sont ainsi les seuls à définir l'hyperplan optimal. C'est pourquoi ils sont parfois appelés "exemples critiques".

Dans le cas non-linéaire la solution consiste à transformer l'espace de représentation de l'échantillon d'apprentissage en un espace de plus grande dimension dans lequel il existe une séparation linéaire. Mais comment trouver cette transformation non-linéaire Φ ? En pratique cela équivaut souvent à connaître la solution d'avance. C'est ici qu'interviennent les fonctions noyaux. Muni de Φ le problème à résoudre serait de trouver les α tels que :

$$\begin{cases} \max\limits_{\alpha} \{\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j u_i u_j (\Phi(x_i).\Phi(x_j))\} \\ \alpha_i \geq 0, i = 1, ..., m \\ \sum_{i=1}^{m} \alpha_i u_i = 0 \end{cases}$$

On remarque que Φ n'intervient que dans le produit scalaire $\Phi(x_i).\Phi(x_j)$. Plutôt que de trouver Φ, on peut donc chercher à trouver la fonction k telle que :

$$k(x, x') = \Phi(x).\Phi(x')$$

Cette fonction k est une fonction noyau. Elle permet, lorsqu'elle est bien choisie, d'utiliser des représentations non-vectorielles et d'éviter de calculer la représentation des exemples dans le nouvel espace. Plusieurs noyaux sont couramment utilisés et parfois combinés : noyau linéaire, polynomial, gaussien ou laplacien (Schölkopf et Smola 2002).

Enfin, il est souvent nécessaire d'assouplir les contraintes, par exemple afin de gérer le bruit sur l'échantillon d'apprentissage. Pour cela, on introduit des *variables ressorts*. L'utilisateur doit alors fixer une constante pour régler le compromis entre la maximisation de la marge et les erreurs de classification.

S'appuyant sur les méthodes d'optimisation, les SVM permettent de traiter des données de grandes dimensions et donnent de bons résultats en pratique. En outre, elles offrent de par leur origine de bonnes garanties théoriques mais demeurent difficiles à mettre en place sur des cas réels.

2.2.1.4 Algorithmes génétiques

Développés par Holland 1975, les algorithmes génétiques sont une technique d'optimisation imitant de manière très simplifiée l'évolution des espèces au moyen de la sélection naturelle. L'adaptation ainsi obtenue peut être considérée comme le fruit d'un mécanisme d'apprentissage. Nous exposons ici les principes de base des algorithmes génétiques ainsi que leur application en apprentissage supervisé.

La première condition dans l'application d'un algorithme évolutionnaire (autre nom des algorithmes génétiques) est de disposer d'un découplage de l'espace des hypothèses. Il faut un espace *phénotypique* dans lequel les hypothèses peuvent être évaluées, et un espace *génotypique* dans lequel elles sont manipulables et transformables par des opérateurs spécifiques. Une hypothèse est donc représentée par son *génotype* qui est une chaîne de valeurs (par exemple une chaîne de bits). L'idée est ensuite de créer une population initiale d'hypothèses puis d'en croiser les individus pour faire évoluer leur génotype de génération en génération jusqu'à atteindre une hypothèse satisfaisante. Concrètement, la génération initiale est le plus souvent tirée au hasard, puis le processus d'évolution se déroule en quatre étapes répétées jusqu'à la satisfaction d'un critère d'arrêt.

Étape 1 : Évaluation. Évaluer la génération courante constitue la première étape. Cela passe en général par l'évaluation de chaque individu à l'aide d'une fonction particulière que l'on appelle fonction de fitness. Cette fonction s'applique sur un phénotype et retourne un score de performance. Dans le cas de l'apprentissage supervisé il s'agit de tester chaque hypothèse sur un ensemble de validation, mais de nombreuses fonctions d'évaluation sont possibles selon le problème traité et peuvent même prendre la forme de simulations (Jin 2005).

Étape 2 : Sélection. La deuxième étape est la sélection des individus qui seront procréateurs, c'est-à-dire dont les génotypes seront croisés pour obtenir de nouveaux phénotypes que l'on espère plus performants. Cette étape est le pendant artificiel de la pression environnementale. La solution la plus simple est de choisir les n meilleurs individus. Cependant, maintenir une certaine diversité est un avantage pour éviter les minimums locaux. Une solution couramment adoptée est donc d'attribuer une probabilité de sélection à chaque individu qui grandira avec son score à l'évaluation. Ainsi une chance de procréer est laissée à tous les individus tout en favorisant les plus aptes. Pour être moins sensible aux éventuelles erreurs de la fonction de *fitness* et s'épargner un coût de calcul important, une autre solution est la sélection par tournoi où les individus sont comparés par petits groupes dont on garde le meilleur (Filipović 2012).

Étape 3 : Croisement. Une fois les procréateurs sélectionnés, on procède à leur croisement (cross-over). Leurs génotypes sont alors mélangés à l'aide d'un *opérateur de croisement* qui produit deux nouveaux génotypes inédits à partir d'une paire de génotypes parents. Par exemple, un opérateur classique dit "croisement à un point" tire un point au hasard dans un génotype et intervertit les gènes des parents entre ce point et le bout de la chaîne. Une variante existe en sélectionnant deux points, comme illustré dans la figure 2.7. En croisant ainsi des individus supposés parmi les meilleurs, on espère aboutir à un nouvel individu combinant les avantages de ses deux parents. Si ce n'est pas le cas, il ne sera sans doute pas gardé lors de la prochaine étape de sélection. Au mécanisme de croisement s'ajoute une probabilité de mutation définie pour chaque gène. Celui-ci peut changer aléatoirement de valeur à chaque fois qu'un nouvel individu est généré. Cela peut permettre l'apparition de nouvelles propriétés phénotypiques qui seront gardées par le processus de sélection si

elles sont profitables, et jetées sinon. Toutefois, si ce taux de mutation est trop important, la conservation des bonnes propriétés déjà acquises est mise en péril.

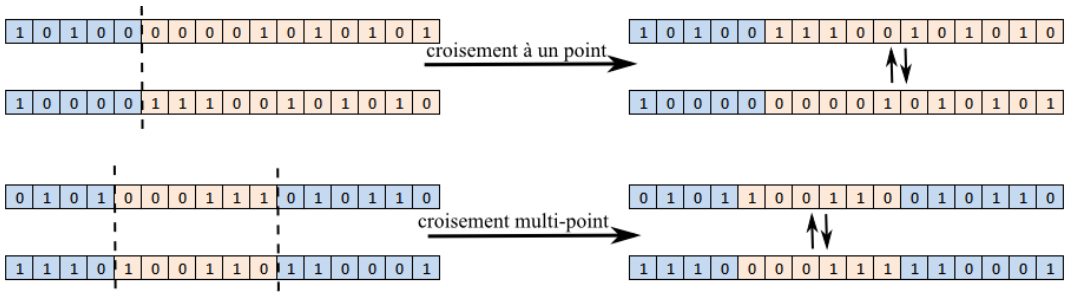

Figure 2.7 – Opérateurs de croisement pour les algorithmes génétiques.

Étape 4 : Remplacement. Enfin, la dernière étape consiste à remplacer la population courante par la nouvelle génération obtenue par croisement et mutation. Ces quatre étapes sont exécutées en boucle, créant des nouvelles générations dont les meilleurs individus sont de plus en plus adaptés, c'est-à-dire dont les scores sont de plus en plus élevés en regard de la fonction d'évaluation. L'algorithme se termine lorsqu'un critère d'arrêt dépendant du problème est atteint. Dans le cas de l'apprentissage supervisé, cela peut par exemple être la classification sans erreur d'un échantillon de validation par une hypothèse de la génération courante.

Parmi les spécialisations des algorithmes génétiques, on compte l'évolution d'automates qui consiste à utiliser des automates à états finis comme génotypes et d'utiliser des opérations sur les graphes comme opérateurs de mutation, les croisements étant abandonnés (Kauffman et Smith 1986). Plus connue, la programmation génétique (*Genetic Programming*) propose de produire des programmes à l'aide d'un algorithme génétique. Cela est possible notamment en représentant les programmes sous forme d'arbres en guise de génotypes. Si la grammaire du langage de programmation cible est suffisamment simple, il est possible de s'assurer que les opérations de croisement et de mutation sur un arbre conservent la validité du programme correspondant (McKay et al. 2010).

Une première limite importante de cette approche concerne son applicabilité. En effet, le choix de paramètres comme la taille et la composition de la population initiale, ou encore l'encodage des objets en génotype ne sont pas triviaux. De même, choisir l'opérateur de croisement et le taux de mutation est un enjeu crucial pour la performance de l'algorithme et il est très difficile de le faire a priori. C'est pourquoi les recherches se sont concentrées sur des mécanismes d'adaptation de ces paramètres en cours d'exécution (Kramer 2010). La technique la plus connue, CMA-ES, s'appuie sur l'adaptation de la matrice de co-variance de la distribution de probabilité régissant la mutation (Auger et Hansen 2005). Dans le même ordre d'idée, on parle de *coévolution* lorsque la fonction de *fitness* change au cours du temps voire diffère pour chaque individu ou groupe d'individus. Cette approche est utile lorsqu'on cherche à décomposer un problème en parties (Potter et De Jong 2000).

Enfin, pour donner de bons résultats, les algorithmes génétiques ont, entre autres, besoin d'une population suffisamment grande et d'un nombre de générations important. Aussi, ils sont rapidement coûteux en terme de calcul.

2.2.1.5 Réseaux de neurones artificiels

Introduits pour la première fois par McCulloch et Pitts 1943, les réseaux de neurones (parfois appelés réseaux connexionnistes) sont un des piliers de l'intelligence artificielle et de l'apprentissage automatique. Étant très étudiés et présents dans de nombreuses applications, nous nous limitons ici à leur utilisation en apprentissage supervisé, en insistant particulièrement sur une forme particulière que sont les *perceptrons*. Ce type de réseau, d'abord proposé par Rosenblatt 1957 puis étendu à une architecture multi-couche (Fiesler 1996), est en effet bien adapté aux problèmes de classification typiques de l'apprentissage supervisé.

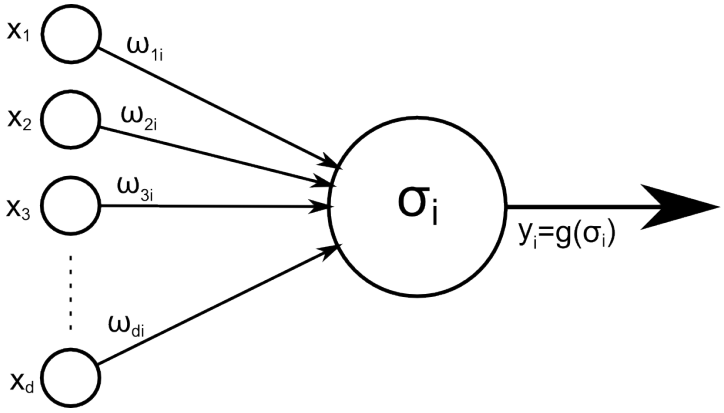

Figure 2.8 – Modèle d'un neurone formel.

La figure 2.8 représente l'unité de base du traitement de l'information dans un réseau connexionniste : le neurone formel. Celui-ci dispose d'un ensemble d'entrées appelé source et d'une sortie. Il est représenté par son état σ_i (i étant l'indice du neurone dans le réseau) et par sa fonction de sortie g avec laquelle il calcule sa valeur de sortie y_i :

$$y_i = g(\sigma_i) \tag{2.5}$$

Le plus souvent, la fonction g est soit la fonction signe, soit une fonction sigmoïde (c'est-à-dire une fonction en forme de s), de formule :

$$g(x) = \frac{1}{1 + e^{-\lambda x}} \tag{2.6}$$

Chaque entrée j d'un neurone i est associée à un poids ω_{ji}. La valeur de σ_i est calculée à partir des d valeurs x_j transmises sur les entrées et de leur poids associé ω_{ji}, en général avec la formule simple :

$$\sigma_i = \sum_{j=1}^{d} \omega_{ji} x_j \tag{2.7}$$

Toutes les architectures sont imaginables selon la tâche que l'on demande au réseau. Nous nous intéressons ici aux *perceptrons* multicouches dans lesquels les neurones formels se classent en trois catégories :

- Les *neurones d'entrées* servent à transmettre les données d'entrées (les exemples de l'échantillon d'apprentissage aussi bien que les futurs exemples à classer). Dans ce cas particulier, $\sigma_i = x_i$ où x_i est la composante d'indice i du vecteur de données x.
- En bout de chaîne, les *neurones de sortie* sont ceux qui fournissent l'hypothèse d'apprentissage. Chaque neurone de sortie correspond à une classe.
- Les *neurones cachés* sont exclusivement connectés à d'autres neurones et non aux entrées/sorties du réseau. Ils effectuent des traitements intermédiaires.

Les *perceptrons* sont construits selon la logique du *feed-forward*, c'est-à-dire que l'information ne se déplace que dans un sens, des neurones d'entrée vers les neurones cachés puis vers les neurones de sortie. Ils sont ainsi constitués en couches successives (couche d'entrée, couches cachées et couche de sortie) dont chaque élément est connecté à tous les éléments de la couche suivante et uniquement à ceux-là. Les neurones d'entrée sont activés en recevant chacun une composante du vecteur x (la donnée d'entrée). Ils effectuent le calcul de leur valeur de sortie (équations 2.5 et 2.7), puis le résultat est transmis à la première couche cachée qui fait de même, et ainsi de suite jusqu'à arriver à la couche de sortie. Le résultat est alors la classe correspondant au neurone de sortie ayant la valeur de sortie la plus grande. Le nombre de couches cachées et de neurones dans ces couches est à définir par le concepteur et influera sur la complexité des frontières qui seront trouvées par le réseau.

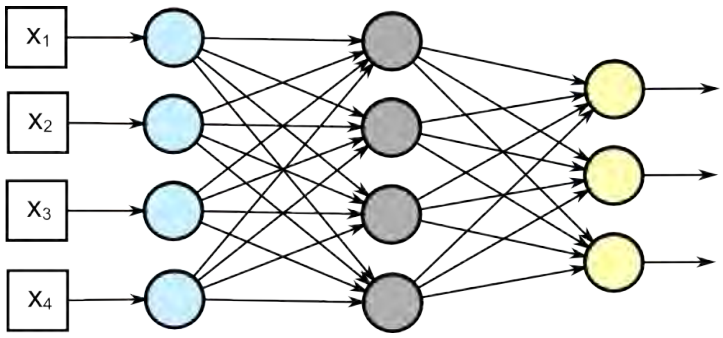

Figure 2.9 – Exemple de perceptron à une couche cachée.

Un exemple simple d'un tel réseau est donné par la figure 2.9. Les entrées sont représentées par des carrés et les neurones par des cercles (bleu pour les entrées, gris pour les cachés et jaune pour les sorties). Il s'agit d'un réseau à une couche cachée pour un problème à trois classes avec des données de dimension 4.

Seul, un réseau de neurones n'est pas capable d'apprentissage. Il faut l'agrémenter d'un mécanisme de rétropropagation de l'erreur entre sortie désirée et sortie calculée qui lui permette d'ajuster le poids des connexions, et donc d'apprendre la fonction qu'il doit exécuter (Rumelhart, Hinton et Williams 1986). Les données de l'échantillon d'apprentissage sont passées séquentiellement au réseau qui va ajuster ses poids à chaque passage. Pour permettre

au réseau de converger, il faut répéter ce processus un grand nombre de fois (en général une centaine de passages pour chaque exemple). L'ajustement des poids est dicté par ce que l'on appelle la *règle delta*, qui exprime la modification à appliquer au poids de la liaison entre le neurone i et le neurone j :

$$\Delta\omega_{ij} = \alpha\delta_j y_i$$

où y_i est la sortie du neurone i (une des entrées du neurone j), α est un coefficient positif entre 0 et 1, et δ_j est une valeur qui dépend de l'erreur. Pour un neurone de sortie, elle s'exprime de la manière suivante (en supposant que la fonction g choisie est une sigmoïde paramétrée avec $\lambda = 1$) :

$$\delta_j = (u_j - y_j)y_j(1 - y_j)$$

où u_j est la sortie désirée (correspondant à l'étiquette de l'exemple passé en entrée du réseau), et y_j la sortie calculée. On obtient récursivement la valeur de δ pour les neurones cachés :

$$\delta_j = y_j(1 - y_j) \sum_{k \in dest(j)} \delta_k \omega_{jk}$$

où $dest(j)$ est l'ensemble des neurones recevant la sortie de j. Le calcul de δ provient directement de la dérivation de la fonction de sortie des neurones. C'est pourquoi cette fonction doit être différentiable et non linéaire. En outre, on remarque que l'ajustement des poids d'une couche n'est possible que si celui de la couche inférieure a été effectué. Cette méthode d'ajustement est ainsi appelée *règle de rétropropagation du gradient de l'erreur*.

L'apprentissage de réseaux connexionnistes multicouches donne de bons résultats malgré une certaine lenteur de l'apprentissage. Un des problèmes ouverts concerne l'arrêt de l'apprentissage puisque le réseau ne se stabilise jamais complètement. La solution apportée est généralement l'utilisation d'un ensemble d'exemples de validation. Un autre problème provient de l'initialisation des poids, certaines conditions initiales pouvant mener à un minimum local.

D'autres types de réseaux sont possibles, notamment ceux présentant des cycles. On parle alors de réseaux connexionnistes récurrents. Cependant de tels réseaux sont difficiles à comprendre et trouver une règle permettant d'ajuster les poids pour parvenir à un apprentissage devient ardu. Parmi ceux utilisés en mode supervisé citons les réseaux connexionnistes à réservoir (*reservoir computing*, Lukoševičius et Jaeger 2009). Découverts simultanément sous les appellations différentes de *Echo State Networks* (Jaeger 2001) et *Liquid States Machines* (Maass, Natschläger et Markram 2002) l'idée derrière ce type de réseaux est de regrouper la composante récurrente (celle présentant des cycles) dans une partie à part du réseau, le réservoir. L'apprentissage est alors uniquement effectué sur un réseau linéaire à une couche dont les entrées sont des neurones choisis dans le réservoir. Cela se rapproche de l'idée de redescription des données d'entrées que l'on a vue avec les SVM (voir 2.2.1.3). Cette approche est encore récente et peu explorée. Cependant, certaines caractéristiques nécessaires aux réservoirs semblent se détacher. Ils doivent être suffisamment grands et posséder des connexions clairsemées pour pouvoir présenter une dynamique riche. Cela signifie des réservoirs de dizaines de milliers de neurones avec un faible taux de connexion (inférieur à 20%) dont

les poids sont tirés aléatoirement. Néanmoins, il reste encore à comprendre comment mieux adapter les réservoirs aux situations d'apprentissage concrètes.

Cela nous amène à une question cruciale des réseaux de neurones artificiels : comment choisir l'architecture d'un réseau ? Une réponse provient de l'utilisation des algorithmes génétiques (voir 2.2.1.4) avec la *neuro-évolution*. Il s'agit de représenter un réseau de neurones (aussi bien sa topologie que ses poids) sous la forme d'un génotype et de le faire évoluer jusqu'à arriver à une solution acceptable. Il existe plusieurs techniques qui diffèrent notamment par leur méthode d'encodage des réseaux en gènes et par la variante d'algorithme génétique utilisée et qui partagent leur coût très important (Floreano, Dürr et Mattiussi 2008).

Les réseaux de neurones sont encore très étudiés, et si les classiques multicouches semblent avoir livré la plus grande partie de leurs secrets, les réseaux récurrents sont encore à défricher. De manière générale, les réseaux connexionnistes donnent de bons résultats, mais sont limités par le temps que nécessite l'apprentissage ainsi que par la difficulté que représente leur mise en œuvre sur un problème particulier (choix de la topologie, etc). Néanmoins ils présentent l'intérêt de promouvoir l'idée qu'un ensemble d'entités simples mais judicieusement connectées entre elles peuvent accomplir des tâches complexes. Enfin, les réseaux de neurones sont souvent considérés comme un cas particulier des réseaux bayésiens que nous présentons dans la prochaine section (Griffiths et Yuille 2008).

2.2.1.6 Apprentissage de réseaux baysésiens

Les réseaux d'inférence bayésiens sont des modèles probabilistes permettant de raisonner à partir de données incertaines. Formulés sous la forme de graphes orientés acycliques, ils contiennent une représentation des connaissances dont ils sont capables de calculer les probabilités conditionnelles (Pearl 1985). Chaque nœud correspond à une variable aléatoire et contient une table de probabilités conditionnelles tandis que chaque arc exprime une relation de dépendance directe.

La figure 2.10 montre un exemple tiré de (Ben-Gal 2007) décrivant la situation d'une personne pouvant avoir une blessure au dos (notée B) et éventuellement ressentir des douleurs (D). Cette blessure peut être le résultat d'une pratique sportive inadéquate (S) ou d'une chaise inconfortable au travail (C). Dans ce dernier cas, on peut supposer qu'un collègue de travail présente également une blessure au dos (E). Toutes ces variables sont binaires et prennent donc les valeurs vrai (V) ou faux (F). Le graphe donne les relations de dépendance entre ces variables à partir desquelles deux types d'inférences peuvent être calculés :

- l'inférence *top-down* lorsque l'on cherche à calculer les probabilités conditionnelles d'un nœud à partir de son ascendance dans le graphe, également appelée *predictive support* en anglais.
- l'inférence *bottom-up* lorsque l'on cherche à calculer les probabilités conditionnelles d'un nœud à partir de sa descendance dans le graphe, également appelée *diagnostic support* en anglais.

Ces inférences sont très coûteuses à calculer, le problème étant connu comme NP-difficile dans le cas général. En revanche, il existe des algorithmes efficaces sur des cas particuliers,

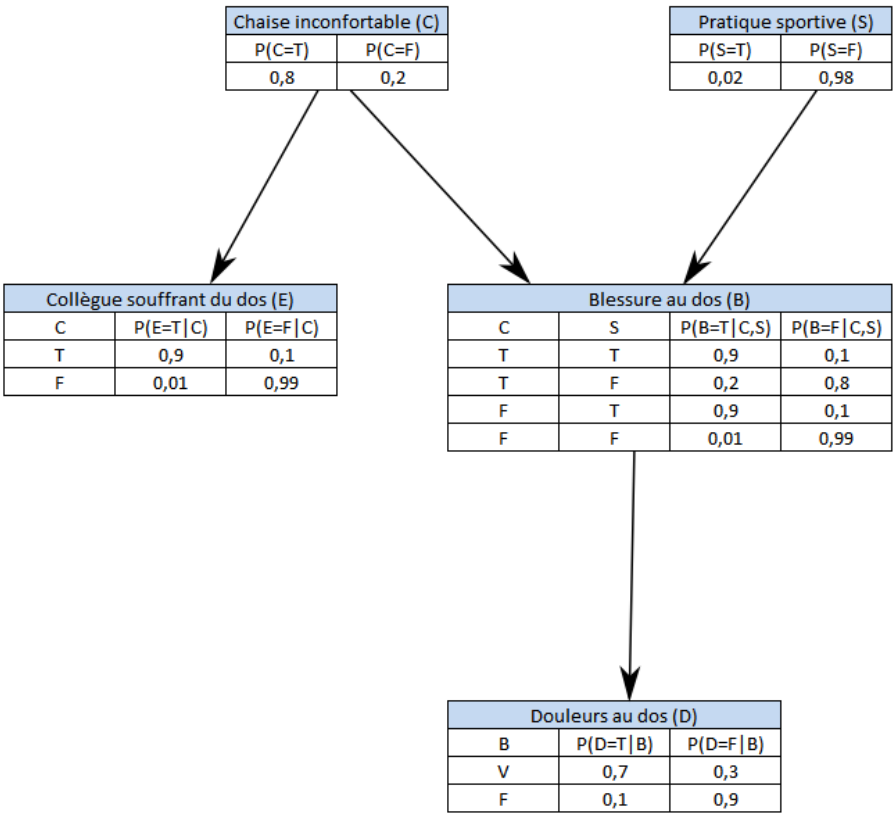

FIGURE 2.10 – Exemple de réseau bayésien.

comme l'algorithme par envoi de messages (LAURITZEN et SPIEGELHALTER 1988) ou celui par élimination de variables (ZHANG et POOLE 1996), ou encore d'autres s'attachant à approximer le résultat à l'aide par exemple de méthodes MCMC (*Markov Chain Monte Carlo*, ALTEKAR et al. 2004).

Ce qui nous intéresse ici est de savoir comment inférer un réseau bayésien à partir d'un échantillon d'apprentissage. Plusieurs familles d'approches se distinguent selon deux critères : la connaissance *a priori* de la topologie du graphe, et la prise en compte d'une observabilité partielle des données (HECKERMAN 2008). Dans le cas le plus simple (structure connue à l'avance), il s'agit de retrouver les bonnes valeurs de probabilités dans les tables à partir des exemples. Les techniques les plus courantes se basent sur le principe de maximum de vraisemblance lorsque toutes les données sont observables, ou bien sur un algorithme de maximisation de l'espérance si ce n'est pas le cas.

Que les données soient entièrement accessibles ou non, apprendre la structure du graphe est extrêmement coûteux. Deux approches se démarquent. D'un côté il y a les méthodes qui attribuent un score à chaque graphe candidat, afin de mesurer la compatibilité entre les dépendances représentées dans le réseau et celles observées dans les données de l'échantillon. Des techniques de recherche dans un espace de possibilités sont ensuite utilisées pour retrouver le meilleur candidat. Citons par exemple (ELIDAN, NACHMAN et FRIEDMAN 2007).

L'inconvénient de cette méthode est que la recherche du meilleur candidat est un problème NP-complet. De l'autre, nous avons des méthodes basées sur des contraintes, moins utilisées. Celles-ci commencent par construire un graphe non-orienté à partir de dépendances repérées grâce à des tests statistiques sur l'échantillon d'apprentissage. Les arcs de ce graphe sont ensuite progressivement orientés par propagation de contraintes. Ici, les principales limites sont l'explosion du nombre de tests à effectuer et la fiabilité du test statistique lorsque l'échantillon ne contient pas assez d'exemples. Des heuristiques sont alors utilisées pour contourner ces difficultés (Steck 2001).

L'avantage des réseaux bayésiens est qu'ils s'adaptent bien aux données incomplètes, ce qui en fait une technique populaire. Leur apprentissage, en particulier lorsque la topologie n'est pas connue à l'avance, demeure cependant très coûteux même dans le cas de petits réseaux. Les applications se concentrent donc sur l'apprentissage des paramètres, laissant à un expert la tâche de modéliser les dépendances entre variables par la construction manuelle d'un graphe.

Avant de faire le bilan de cette partie sur l'apprentissage supervisé, il semble maintenant intéressant d'aborder un ensemble de techniques un peu spéciales visant à améliorer les performances de programmes dits "faibles" en les combinant.

2.2.1.7 Méta-apprentissage

Le méta-apprentissage est l'apprentissage aussi bien du choix entre plusieurs programmes d'apprentissage que de leur combinaison (Vilalta et Drissi 2002). Il peut s'agir d'algorithmes différents ou bien de plusieurs instances du même algorithme ayant bénéficié d'échantillons de données différents. Diverses méthodes ont été proposées, parmi elles nous abordons succinctement ici : l'apprentissage à deux étages, le *bagging*, le dopage et enfin l'apprentissage en cascade.

Faire voter un ensemble de classifieurs est certainement la manière la plus simple de les combiner : la décision finale pour un point d'entrée est la moyenne pondérée de la décision de chacun des programmes d'apprentissage. La question est alors de savoir quel poids accorder à chaque vote. C'est l'objet de l'apprentissage à deux étages (Wolpert 1992). Cette technique utilise un classifieur particulier (appelé méta-classifieur) chargé d'apprendre la bonne valeur des poids des votes de chaque classifieur élémentaire. Elle consiste à séparer l'échantillon d'apprentissage S en deux parties S_1 et S_2 et à faire apprendre les classifieurs élémentaires uniquement sur S_1 pour ensuite les tester sur S_2. Les paramètres du méta-classifieur sont alors appris en se basant sur un nouvel échantillon formé des résultats des tests sur S_2 étiquetés par la sortie réelle désirée.

Le *bagging* (contraction de *bootstrap aggregation*) utilise plusieurs instances d'un même algorithme. Chacune d'elle est entraînée sur une fraction de l'échantillon d'apprentissage tirée aléatoirement et avec remise. Pour une entrée donnée, à chaque tirage va donc correspondre une hypothèse et l'hypothèse finale est tout simplement leur moyenne (Breiman 1996).

Le dopage, plus connu sous le nom anglais de *boosting*, regroupe les techniques produisant des décisions précises à partir de décideurs dits faibles (c'est-à-dire que leurs performances

sont à peine meilleures que le hasard). Le premier algorithme a avoir été proposé est celui des sous-ensembles. Une première hypothèse est obtenue sur un sous-échantillon S_1. On apprend une deuxième hypothèse sur un autre sous-échantillon S_2 issu de $S - S_1$ et dont la moitié des données sont mal classées par la première hypothèse. Enfin, on apprend sur un troisième sous-échantillon tiré dans $S - S_1 - S_2$ et pour lequel les deux premières hypothèses sont en désaccord. L'hypothèse finale est alors le résultat d'un vote entre les trois hypothèses apprises (Schapire 1990). On peut appliquer récursivement cet algorithme et utiliser ainsi neuf sous-ensembles, puis ving-sept, et ainsi de suite. AdaBoost est une généralisation probabiliste du *boosting* par sous-ensembles. Elle est basée sur l'idée d'appliquer une distribution de probabilités sur les exemples de l'échantillon d'apprentissage en fonction des résultats des hypothèses précédentes. Ces probabilités sont ensuite utilisées pour former les sous-échantillons (Freund et Schapire 1997). Il existe plusieurs variations de cet algorithme, comme BrownBoost (Freund 2001), Gentle AdaBoost (Hastie et al. 2005) et Multi-class AdaBoost (Zhu et al. 2009).

Contrairement au dopage et au *bagging*, l'apprentissage en cascade (*cascading* en anglais) met les classifieurs élémentaires en série. Un exemple à classer est donné au premier, puis le résultat est évalué. S'il est considéré comme fiable, l'algorithme s'arrête. Sinon, on donne l'exemple au second classifieur, et ainsi de suite. L'avantage de cette méthode est d'optimiser le temps de calcul en plaçant en premier les classifieurs les plus simples. Les plus coûteux ne seront alors appelés qu'en cas d'exemple difficile. En revanche, elle impose de disposer d'une fonction d'évaluation pour chaque type de classifieur présent dans la chaîne (Kaynak et Alpaydin 2000).

L'idée derrière le méta-apprentissage, celle de parvenir à faire mieux que la performance individuelle des programmes en les combinant judicieusement est particulièrement intéressante, nous en retrouverons un équivalent dans le chapitre 3.

2.2.1.8 Bilan de l'apprentissage supervisé

D'autres techniques d'apprentissage supervisé existent, notamment la classification naïve bayésienne (Domingos et Pazzani 1997) et les forêts d'arbres décisionnels (qui sont une extension du *bagging* appliqué aux arbres de décision, Breiman 2001), mais ce bref aperçu du domaine suffit à en percevoir les limites quant au problème qui nous intéresse. En effet, le besoin d'un échantillon d'apprentissage réserve cette approche aux problèmes statiques dont on a une idée précise de la solution. Cela ne paraît donc pas adapté au nôtre, pour lequel la solution n'est pas connue à l'avance. En outre, si les techniques présentées ici sont applicables très naturellement pour des problèmes de classification, adapter le problème du contrôle à ces techniques n'est pas une mince affaire. Néanmoins, ce tour d'horizon nous a permis d'introduire les concepts de base de l'apprentissage ainsi que certaines des techniques appliquées dans le cadre du contrôle intelligent. Nous allons maintenant voir que des approches comme les neurones artificiels sont également utilisables avec un fonctionnement non supervisé, que nous présentons ci-après.

2.2.2 Apprentissage non supervisé

À la différence des méthodes présentées jusque là, celles d'apprentissage non supervisé ne bénéficient pas d'un oracle pour les guider. Elles ne reposent ni sur une fonction d'évaluation, ni sur l'étiquetage d'un échantillon d'exemples. Leur but est d'apprendre quelles sont les séparations, les corrélations ou encore les composantes naturelles au sein de données non étiquetées. Les algorithmes les plus classiques sont ceux des *k*-moyennes ainsi que les analyses par composantes, mais de nombreuses autres techniques utilisent les réseaux de neurones. Nous présentons parmi celles-ci les cartes de Kohonen, les réseaux de Hopfield et les machines de Boltzmann.

Mais avant d'aborder les techniques d'apprentissage non supervisé, nous décrivons brièvement ci-après la classe des algorithmes d'apprentissage semi-supervisés, qui se basent sur un échantillon partiellement étiqueté.

2.2.2.1 Apprentissage semi-supervisé

On regroupe sous le terme d'apprentissage semi-supervisé les techniques prenant en compte l'entièreté d'un échantillon partiellement étiqueté. On a donc l'échantillon d'apprentissage S composé d'un échantillon supervisé S_{sup} et d'un non supervisé S_{nsup} :

$$S = S_{sup} \cup S_{nsup}$$

Le co-apprentissage en est une approche classique. Il repose sur la disponibilité de deux représentations de chaque donnée de l'échantillon, par exemple une page web peut être décrite à la fois par ses hyperliens et par les mots qu'elle contient. La technique consiste à d'abord classer les exemples de S_{nsup} grâce à un apprentissage supervisé basé sur S_{sup}. Cet apprentissage est réalisé deux fois : une fois par le classifieur A sur la première représentation des données, et une deuxième par le classifieur B sur la deuxième représentation des données. On garde ensuite de ces deux apprentissages les p exemples les plus sûrs, on les ajoute à S_{sup} et on réitère jusqu'à la convergence (Blum et Mitchell 1998). De cette manière, A et B s'ajoutent mutuellement des connaissances, puisqu'un exemple sûr pour l'un ne l'est pas forcément pour l'autre.

D'autres méthodes existent, comme les SVM transductifs semi-supervisés (S3VM, Bennett et Demiriz 1999), mais toutes ne sont intéressantes que si l'on dispose d'un étiquetage d'une part, et que l'on connait les exemples sur lesquels on va être interrogé d'autre part.

2.2.2.2 Algorithme des k-moyennes

L'algorithme des *k*-moyennes (*k-means* en anglais) cherche à partitionner les exemples d'un échantillon en k classes. Il repose sur l'idée simple de minimiser la variance au sein de chaque partition et est constitué de deux étapes exécutées en boucle : l'allocation et le recentrage (Hartigan et Wong 1979).

Pour amorcer l'algorithme, k exemples sont sélectionnés au hasard dans l'échantillon. Ils sont parfois appelés graines (*seeds*) et servent de centres de gravité provisoires. La phase

d'allocation consiste à associer chaque autre exemple à la classe correspondant à la graine la plus proche (selon une fonction de distance propre au domaine de l'échantillon). Les centres de gravité (les points moyens) des partitions ainsi formées sont ensuite calculés et les exemples de l'échantillon réalloués en fonction de ceux-ci, formant de nouvelles classes. Puis on calcule à nouveau les centres de gravité, et ainsi de suite jusqu'à ce que les partitions (et donc les centres de gravité) soient stables d'une passe à l'autre.

On observe que la somme des variances des classes diminue à chaque passe jusqu'à la stabilisation de l'algorithme. Cependant, rien ne garantit que le partionnement trouvé soit le meilleur. En effet, il se peut qu'en sélectionnant des points de départs différents les partitions trouvées soient différentes et la somme de leur variance encore plus basse. Pour minimiser ce problème, l'algorithme des *k*-moyennes++ propose une méthode de choix des graines, basée sur l'intuition qu'il est préférable qu'elles soient uniformément réparties sur l'échantillon (Arthur et Vassilvitskii 2007).

Étant largement répandu et étudié, l'algorithme des *k*-moyennes a vu de nombreuses variantes apparaître comme les *k*-médianes (Juan et Vidal 2000) ou les *k*-médoïdes (Park et Jun 2009) qui utilisent d'autres manières de calculer les centres de gravité. De nombreuses autres variantes cherchent quant à elles à optimiser les lourdes exigences calculatoires des *k*-moyennes (Kanungo et al. 2002, Elkan 2003).

Outre son coût, un frein important à l'utilisation de cette technique sur des cas concrets est le choix de *k* par l'utilisateur. Ce choix revient à faire le compromis entre variance et nombre de classes, ce qui demande une certaine connaissance du domaine qui n'est pas forcément disponible lorsqu'on envisage de faire de l'apprentissage non supervisé.

2.2.2.3 Analyse en composantes

À l'image des k-moyennes, les méthodes d'analyse par composantes ont pour but de dégager des sous-ensembles cohérents d'un échantillon de données. Elles cherchent à représenter les données dans un espace où les différentes variables qui les constituent ne sont pas corrélées. C'est cette représentation qui est apprise à partir d'un échantillon.

Par exemple, l'analyse en composantes principales (ACP) est une technique classique qui réduit la dimension des données de manière à ce que les variables corrélées soient confondues en une seule nouvelle variable. Ainsi, elle crée un nouvel espace de représentation dont les axes correspondent à des variables indépendantes. À partir d'un échantillon d'apprentissage S contenant m vecteurs x, on calcule la moyenne μ et la matrice de covariance Σ :

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu)(x_i - \mu)^\top$$

Les vecteurs propres associés aux valeurs propres les plus grandes de Σ rendent compte de la corrélation entre les variables. On définit alors la matrice de projection P avec ces vecteurs propres et on obtient la représentation de x dans le nouvel espace projeté avec :

$$x_P = P^\top (x - \mu)$$

Cette méthode apprend donc à partir d'exemples non étiquetés quelles en sont les directions principales, elle se situe à cheval entre l'apprentissage et la fouille de données. Sa principale limite est qu'elle devient inefficace lorsque les données possèdent un très grand nombre de dimensions. La réduction n'est pas significative et on perd la structure naturelle des données. L'apprentissage multi-linéaire de sous-espaces (*multilinear subspace learning*, MSL) est une généralisation de l'ACP qui permet d'éviter ces problèmes en faisant appel à la notion de tenseurs (Lu, Plataniotis et Venetsanopoulos 2011). Elle est par exemple régulièrement utilisée en reconnaissance de formes. Cependant, elle est sensible aux conditions initiales et tombe dans des minimums locaux.

Il existe d'autres types d'analyses en composantes. Par exemple, l'analyse en composantes indépendantes cherche à retrouver des signaux indépendants d'après une combinaison linéaire (Comon et Jutten 2010). Mais elles partagent l'inconvénient de la linéarité de leurs résultats. Une ACP sur des données en deux dimensions donnera toujours une droite de régression, jamais une courbe.

Les réseaux de neurones permettent de contourner cette limitation. Ainsi, les cartes de Kohonen peuvent être vues comme une ACP non linéaire.

2.2.2.4 Cartes de Kohonen

Les cartes de Kohonen sont un type particulier de réseau de neurones visant à produire une représentation simplifiée d'un échantillon de données, en d'autres termes à apprendre les caractéristiques principales d'un ensemble de points. Le principe est de partir d'une grille de neurones (généralement de dimension trois ou inférieure) dont le poids de chaque liaison représente la distance avec le voisin correspondant. Les poids sont ajustés à partir des données, déplaçant ainsi les nœuds de manière à ce que leur placement corresponde à la répartition des exemples de l'échantillon (Kohonen 2001).

Concrètement, on commence par initialiser aléatoirement les poids de la grille. On tire au hasard un exemple de l'échantillon et on détermine le neurone qui lui est le plus proche. Ce neurone (ainsi que chacun de ses voisins, dans une moindre mesure) subit un déplacement le rapprochant encore plus de l'exemple traité. On tire ensuite un nouvel exemple et on recommence, chaque exemple pouvant (et devant) passer plusieurs fois. L'algorithme se termine lorsqu'un critère d'arrêt est atteint. Ce critère est souvent relatif au nombre de passes et est choisi de façon à ce que chaque exemple de l'échantillon soit utilisé une centaine de fois.

La figure 2.11 illustre cet algorithme avec un réseau en deux dimensions. Les neurones sont les intersections de la grille noire. La tâche bleue représente les données de l'échantillon d'apprentissage et le point vert est l'exemple traité. Le neurone le plus proche est entouré en jaune. Il subit un rapprochement important tandis que celui de ses voisins est plus limité. À la fin, la carte recouvre les données.

Après convergence de la carte, la densité locale des neurones traduit celle des données de l'échantillon. Aussi les cartes de Kohonen ne permettent pas réellement de faire de la classification, mais plutôt d'approximer et de représenter sous forme plus compacte des

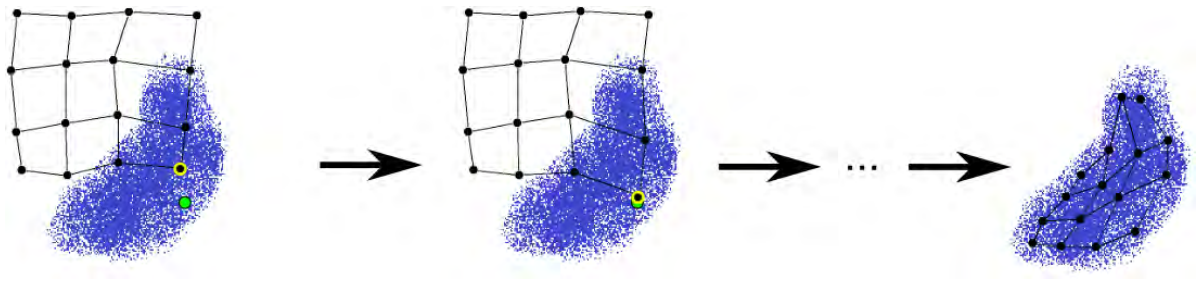

FIGURE 2.11 – Illustration de la convergence d'une carte de Kohonen.

données de grande dimension. En ce sens, elles peuvent être vues comme une ACP non linéaire. Choisir la bonne taille de la carte est un problème difficile auquel une solution proposée est de commencer avec un nombre minimal de neurones pour en ajouter progressivement (ALAHAKOON, HALGAMUGE et SRINIVASAN 2000). Enfin, un grand nombre de passes est nécessaire avant d'arriver à une convergence satisfaisante de la carte. Cela signifie qu'il n'est pas envisageable de parvenir à un apprentissage en temps réel.

2.2.2.5 Réseaux de Hopfield

Un réseau de Hopfield (HOPFIELD 1982) est un réseau connexionniste capable de mémoriser des formes et de les reproduire. Après entraînement, le réseau converge vers une forme mémorisée lorsqu'il est stimulé avec une partie de la forme à retrouver.

Dans un réseau de Hopfield, chaque neurone est connecté à tous les autres, et tous les neurones sont à la fois une entrée et une sortie du réseau (figure 2.12). Les connexions ont en outre la contrainte d'être symétriques, ainsi, pour tout i et j, $\omega_{ij} = \omega_{ji}$. Chaque neurone est binaire, c'est-à-dire qu'il ne peut prendre que deux valeurs, par exemple -1 et 1. On peut donc représenter l'état d'un réseau de N neurones par un mot de N bits. Enfin, le temps est discret, l'activité du réseau est rythmée par une horloge.

L'apprentissage d'une forme se fait par l'ajustement du poids des connexions. Il est réalisé classiquement en suivant une règle appelée *loi de Hebb*. Cette règle pousse la connexion de neurones qui sont dans le même état à avoir un poids important, et inversement. Cette règle peut s'écrire comme :

$$\omega_{ij} = \frac{1}{p} \sum_{k=1}^{p} x_i^k x_j^k$$

où p est le nombre d'exemples d'entraînement et x_i^k la valeur de l'entrée du neurone i au k-ième exemple.

Un neurone calcule son état σ_i en fonction de ce qu'il lit sur ses entrées et des poids ω_{ij} de celles-ci de la manière suivante :

$$\sigma_i = \begin{cases} 1 \text{ si } \sum_j \omega_{ij}\sigma_j > \theta_i \\ -1 \text{ sinon} \end{cases}$$

où θ_i est un seuil prédéfini. Grâce à l'apprentissage hebbien, les poids les plus forts correspondent aux connexions provenant de neurones ayant la même activité. Ainsi, après plusieurs mises à jour, l'état tendra à redevenir celui qui a été mémorisé si une partie suffisante des entrées est stimulée avec la forme à retrouver. En effet, un neurone prendra l'état 1 seulement si le poids de sa connexion lui permet de dépasser le seuil d'activation θ_i, ce qui ne survient que si ce neurone était dans l'état 1 en même temps que l'entrée considérée au moment de l'apprentissage.

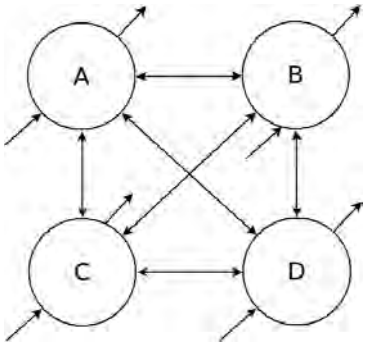

Figure 2.12 – Exemple de réseau de Hopfield à quatre neurones.

Par exemple, on fait apprendre au réseau de la figure 2.12 la forme 1001 (les neurones A et D sont dans l'état 1 et les neurones B et C dans l'état -1). Le poids des liaisons diagonales, et notamment celle entre A et D, est alors grand. Si on place tous les neurones dans l'état -1 et qu'on stimule uniquement A avec la valeur 1, le neurone D va dépasser son seuil et s'activer. En revanche, le faible poids des connexions entre A et les deux autres neurones va les laisser sous le seuil, et donc dans l'état -1. Les neurones A et D sont dans l'état 1 et les autres dans l'état -1 : la forme apprise est retrouvée.

L'état global du réseau peut être représenté par une valeur scalaire appelée *énergie*. Elle est calculée à l'aide de la fonction suivante :

$$E = -\frac{1}{2}\sum_{i,j}\omega_{ij}\sigma_i\sigma_j + \sum_i \theta_i\sigma_i$$

Cette fonction a la propriété de posséder un minimum (éventuellement local) quand le réseau est dans un état correspondant à une forme mémorisée. Ainsi, une diminution de l'énergie dénote une convergence du réseau vers une forme apprise.

Puisqu'ils doivent être stimulés par une version altérée de la forme à retrouver, les réseaux de Hopfield sont particulièrement adéquats pour le filtrage de bruit, la reconnaissance de formes ou encore la reconstruction de formes obstruées. Ils possèdent la propriété importante de toujours converger : lorsque les neurones calculent leur nouvel état, l'énergie n'augmente jamais, elle ne peut que diminuer ou se stabiliser. Cependant, outre le grand nombre de passes nécessaire, ils ont la limitation de ne pouvoir mémoriser que très peu de formes différentes relativement au nombre total de neurones N dans le réseau, soit environ $0,138 \times N$ (Hertz, Krogh et Palmer 1991). Cela est un sérieux frein à leur utilisation dans un cadre pratique.

2.2.2.6 Machines de Boltzmann

Les machines de Boltzmann sont un autre type de réseau de neurones récurrent. Elles sont souvent présentées comme une version stochastique des réseaux de Hopfield. En effet, elles se basent sur des principes probabilistes tout en présentant un certains nombre de ressemblances avec les réseaux de Hopfield comme les neurones binaires, les connexions symétriques, l'apprentissage hebbien et l'utilisation d'une fonction d'énergie (Ackley, Hinton et Sejnowski 1985).

Les machines de Boltzmann disposent de neurones cachés et de neurones visibles. Les neurones visibles sont les seuls à recevoir des informations depuis l'extérieur, ce sont eux qui reçoivent les données d'entrée aussi bien pour l'apprentissage que pour la stimulation. Là aussi, l'apprentissage correspond à l'ajustement du poids des liaisons. Il a lieu en deux étapes qui sont appliquées alternativement :

- la phase positive où la valeur d'un exemple est affectée aux entrées
- la phase négative durant laquelle on laisse le réseau s'exécuter librement

L'ajustement des poids ainsi que le calcul de l'état de chaque neurone sont effectués en utilisant la *distribution de Boltzmann* qui est un outil mathématique provenant de la physique et qui permet d'estimer la fonction de distribution des états d'un système composé d'unités ayant une mesure d'énergie. L'apprentissage correspond à la collecte de statistiques sur l'échantillon de données tandis que la convergence des neurones s'apparente au calcul des probabilités en fonction des statistiques acquises.

En théorie, les machines de Boltzmann peuvent résoudre des problèmes combinatoires complexes et pourraient être suffisamment génériques pour être appliquées à de nombreux cas. Cependant, elles se heurtent à de lourdes difficultés pratiques : elles sont extrêmement sensibles au bruit et leur temps de calcul croit exponentiellement avec la taille de la machine, ce qui exclut leur utilisation sur des problèmes non-triviaux.

Pour contourner en partie ces problèmes, il existe une forme dite "restreinte" des machines de Boltzmann (*restricted Boltzmann machines*, ou RBM, en anglais) dans laquelle les neurones sont organisés en couche à la manière des perceptrons (Hinton 2010). Une couche de neurones visibles est alternée avec une couche de neurones cachés qui sert d'entrée à la couche de neurones visibles suivante, et ainsi de suite. Lorsqu'il y a plus d'une paire couche visible/couche cachée, on parle également de réseau profond de croyance (*deep belief network*)

Si on trouve des exemples d'application de RBM dans la reconnaissance d'images (Hinton, Osindero et Teh 2006), ou encore vocale (Dahl et al. 2010), leur utilisation reste difficile car leur structure (nombre de couches et de neurones) et leur paramétrage (comme le critère d'arrêt de l'apprentissage, le nombre de passes pour chaque exemple, etc) doivent être adaptés à chaque problème. Cela requiert une certaine expertise des machines de Boltzmann, il n'existe pas encore de méthode pour y parvenir facilement. En outre, elles ne sont pas adaptées à un apprentissage en ligne temps-réel.

2.2.2.7 Bilan de l'apprentissage non supervisé

Nous avons vu que les méthodes d'apprentissage non supervisé permettent de classer des données ou d'extraire des associations sans avoir de connaissance préalable sur le résultat attendu. En ce sens, cette classe d'algorithmes parait plus intéressante que celle de l'apprentissage supervisé. En effet, fournir moins de connaissances préalables devrait entraîner moins de travail d'intégration sur un cas concret.

Cependant, les techniques présentées jusqu'ici s'appliquent essentiellement en mode hors-ligne. C'est-à-dire que l'apprentissage se fait en amont, sur un échantillon obtenu suite à une série de mesures, et est ensuite figé. Si l'objet de l'apprentissage évolue au cours du temps, il faut tout reprendre depuis le début. Cela n'a pas empêché certaines approches d'être utilisées en identification de systèmes (voir 2.3) mais cela ne correspond pas non plus aux contraintes d'adaptation en temps réel que l'on se pose.

En revanche, l'apprentissage par renforcement, qui se base sur une mécanique de rétroaction entre l'apprenant et son environnement, semble plus à même de répondre à nos besoins.

2.2.3 Apprentissage par renforcement

L'apprentissage par renforcement s'intéresse au cas particulier où l'apprenant est une entité autonome et permanente dans un environnement dont elle ne connait pas forcément la structure. Celle-ci apprend de ses interactions pour optimiser une certaine fonction d'utilité (parfois également appelée fonction de gain). C'est donc un apprentissage en ligne, basé sur une mesure du gain par rapport à une action effectuée sur l'environnement. Contrairement aux approches précédentes, l'objet de l'apprentissage est ici un comportement, l'association d'états du monde à une action.

Le principal problème à résoudre lorsque l'on cherche une méthode d'apprentissage par renforcement est le dilemme exploration contre exploitation (Cornuéjols et Miclet 2010). En effet, il arrive généralement que soient trouvées des actions bénéfiques après quelques cycles seulement. Il se pose alors la question de savoir s'il est préférable de continuer à explorer ou bien de répéter ces actions pour augmenter le gain au risque de louper de meilleures solutions.

La première approche reprend des idées de la programmation dynamique et des méthodes de Monte-Carlo pour évaluer et améliorer leurs actions dans un environnement modélisable par un processus de décision markovien. Connue sous le nom de méthode des différences temporelles, elle comprend notamment le Q-learning et l'algorithme SARSA présentés dans les paragraphes suivants. Nous abordons ensuite les systèmes de classeurs, qui font appel aux algorithmes génétiques, puis le raisonnement par cas.

2.2.3.1 Q-learning

L'algorithme du Q-learning est un des premiers à avoir été publié. Il se base sur la dépendance entre la fonction d'utilité et les actions pour apprendre un comportement optimal.

Le but est d'estimer correctement la fonction d'utilité $Q(s,a)$ afin que l'action qu'on pense la plus bénéfique le soit effectivement (Watkins et Dayan 1992).

La fonction de gain est estimée à partir de la récompense r (aussi appelée signal de renforcement) obtenue après une action a grâce à une opération itérative dérivée de la méthode de Monte-Carlo :

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a)]$$

où $Q(s,a)$ représente le gain estimé si l'action a est entreprise dans l'état s, $\mathcal{A}$ est l'ensemble des actions, s' est l'état atteint à partir de s avec l'action a, et α et γ sont des paramètres à définir par l'utilisateur.

L'action est ensuite choisie selon une méthode dite ϵ-gloutonne où la meilleure action, selon l'estimation courante de la fonction d'utilité, est sélectionnée avec une probabilité de $1-\epsilon$. Le paramètre ϵ est à choisir entre 0 et 1, et de préférence petit. Plus il est grand, plus l'algorithme a tendance à explorer au lieu de préférer le gain immédiat.

Parce qu'il considère la prochaine action avec le meilleur gain estimé parmi toutes les actions possibles, l'algorithme du Q-learning ne dépend pas d'une quelconque politique d'action. On dit qu'il est "hors politique". Cette méthode garantit de converger (pourvu que tous les états soient visités infiniment souvent) et est relativement facile à mettre en œuvre. Cependant, elle converge moins rapidement que d'autres méthodes car elle nécessite que chaque état de l'environnement soit visité un grand nombre de fois. En outre, il n'existe pas dans le cas général de règle pour en instancier les paramètres. Il faut donc recourir à des règles *ad hoc*.

2.2.3.2 SARSA

L'algorithme SARSA (*State-Action-Reward-State-Action*) est une variante "sur politique" du Q-learning dans laquelle la nouvelle action n'est pas nécessairement choisie grâce à une procédure ϵ-gloutonne mais est sélectionnée selon une certaine politique π au choix de l'utilisateur (Sutton 1996).

Cela implique que la mise à jour itérative de l'estimation de la fonction d'utilité ne dépend plus de la meilleure prochaine action mais de la prochaine action préconisée par la politique suivie π :

$$Q^{\pi}(s,a) \leftarrow Q^{\pi}(s,a) + \alpha[r + \gamma Q^{\pi}(s',a') - Q^{\pi}(s,a)]$$

L'algorithme tire son nom de cette formule utilisant les variables s, a, r, s', a' (l'état courant, l'action courante, le signal de renforcement perçu, l'état suivant et l'action suivante).

L'action suivante est celle définie par π qui se base elle-même sur ses estimations de gain $Q^{\pi}(s,a)$. Cependant, il y a un paramètre ϵ à fixer pour qu'une probabilité existe de changer aléatoirement de politique à des fins d'exploration.

À l'image du Q-learning, SARSA garantit la convergence vers la meilleure politique à condition que le paramètre ϵ soit bien réglé et que les états soient visités infiniment souvent. L'inconvénient est ici le grand nombre de cycles action/renforcement nécessaires

avant une convergence satisfaisante. En outre, Q-learning comme SARSA sont limités aux environnements pouvant être modélisés par un processus décisionnel de Markov, sans quoi l'opération de mise à jour de l'estimation de la fonction de gain n'est pas pertinente. Ils nécessitent également un représentation discrète des états de l'environnement, ce qui en pratique est difficilement faisable.

2.2.3.3 Systèmes de classeurs

Les systèmes de classeurs (*Learning Classifier Systems*, ou LCS, en anglais) sont des outils combinant les principes de l'apprentissage par renforcement aux algorithmes génétiques afin d'apprendre les meilleures interactions possibles avec l'environnement (Buche, Septseault et De Loor 2006).

Un LCS est classiquement constitué d'une mémoire de taille limitée qui stocke le ou les derniers états perçus de l'environnement et d'une base de règles de comportement de type condition-action appelée classeur. Un mécanisme d'appariement entre l'état perçu et les conditions des règles est utilisé pour déclencher ces dernières et activer les actions pouvant être entreprises. Plusieurs règles sont susceptibles d'être déclenchées en même temps, menant donc à un choix à faire entre plusieurs actions préconisées. Ce choix est guidé par une variable appelée "force", associée à chaque règle. Parmi celles qui ont été déclenchées, c'est l'action de la règle la plus forte qui a la plus grande probabilité d'être appliquée. Cette variable est mise à jour à chaque cycle selon la récompense perçue tandis qu'un algorithme génétique s'occupe à la fois de faire apparaître de nouvelles règles et d'éliminer les moins fortes pour ne conserver que les meilleures. La figure 2.13 résume tout ceci.

Un LCS cherche à améliorer la récompense qu'il reçoit de l'environnement. Il exécute de manière répété un cycle perception/comparaison/sélection/action pendant lequel la base de règles est mise à jour simultanément par l'algorithme génétique et par le renforcement. Après un certain nombre de cycles, les actions produites maximisent la récompense reçue.

Il existe de très nombreuses variantes des LCS que l'on peut catégoriser en deux familles selon que l'algorithme génétique agisse sur une population de règles (type Michigan) ou sur une population de bases de règles (type Pittsburgh). Le plus souvent, la fonction d'évaluation de l'algorithme génétique est basée sur la variable force mais des versions existent où c'est la précision de la règle qui est utilisée. On parle alors de XCS (Wilson 1995). Mais un des points cruciaux lors de l'implémentation d'un LCS est de bien choisir le mécanisme de renforcement, celui qui transforme la récompense en modification de la force des règles. En effet, la même action pouvant être proposée par plusieurs règles simultanément, un algorithme tel que le Q-learning n'est pas suffisant. Une solution courante s'appelle la *Bucket Brigade* et propose d'assimiler environnement et règles à des agents financiers dans un système d'enchères (Wilson 1994). Enfin, d'autres variantes incorporent d'autres techniques d'intelligence artificielle, comme les réseaux de neurones et la logique floue (Bull et O'Hara 2002). Le lecteur peut se reporter à Urbanowicz et Moore 2009 pour une description plus exhaustive de la constellation des LCS existants.

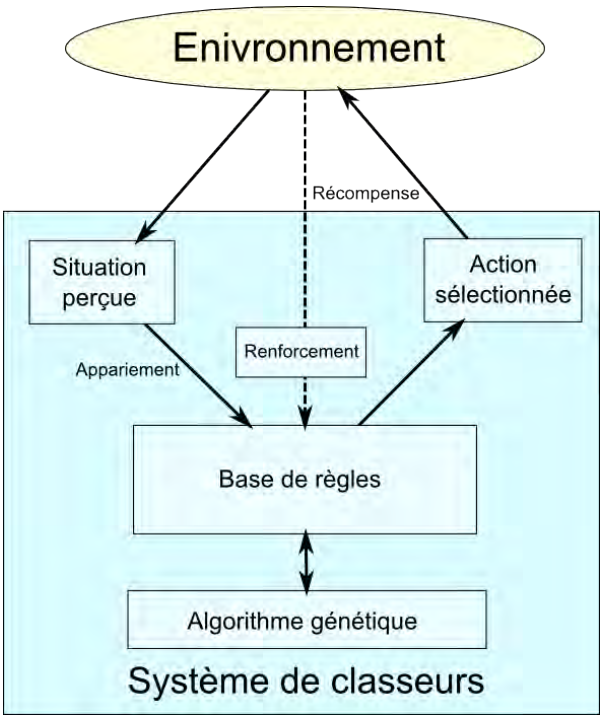

FIGURE 2.13 – Structure de base d'un système de classeurs.

Les LCS ont été appliqués dans de nombreux domaines, comme la sélection d'actions d'un personnage humain dans un appartement virtuel (SANCHEZ 2004) ou encore la reconnaissance audio-visuelle d'émotions (GLODEK et al. 2011). Aussi, ils paraissent adaptés pour une large gamme de problèmes, particulièrement lorsque l'apprentissage concerne une entité permanente qui doit s'adapter sur le long terme. Cependant la nature des algorithmes impliqués dans les LCS fait qu'il n'en existe pas d'idéal, capable d'apprendre et de s'adapter à tout type d'environnement. Parmi les très nombreuses et diverses versions, chacune est adaptée à un cas précis et n'entreprend pas d'être une solution globale pour tous les problèmes (SANZA 2001). En outre, il n'existe pas de critère formel de sélection pour choisir quelle variante utiliser sur un problème concret, il faut se reposer sur l'expérience.

2.2.3.4 Raisonnement par cas

Le raisonnement par cas, ou *Case Based Reasoning* (CBR) en anglais, est une méthode pour résoudre des problèmes en s'inspirant de cas analogues précédemment rencontrés et stockés dans une base de connaissances. La solution extraite de la base est adaptée au problème courant, puis la base est mise à jour avec ce nouveau cas. Un cas se compose d'un problème, d'une solution, et d'indications sur la manière dont la solution a été trouvée et sur les raisons possibles d'un échec. Le processus se décompose en quatre étapes (AAMODT et PLAZA 1994) décrites ci-après et schématisées par la figure 2.14 :

– Rechercher un cas similaire dans la base de connaissance.

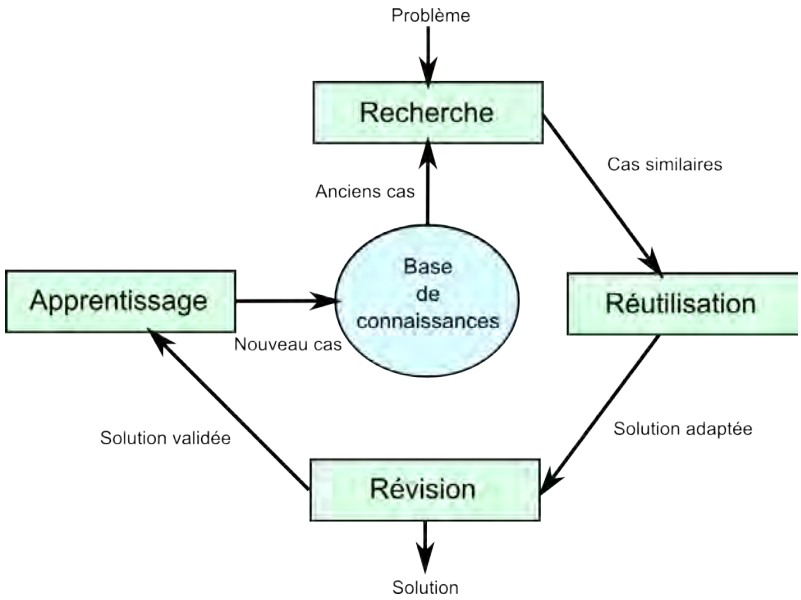

Figure 2.14 – Principales étapes d'un algorithme de raisonnement par cas.

- Réutiliser la solution de ce cas. Cela comprend une éventuelle adaptation de la solution extraite de la base, grâce aux indications accompagnant le cas.
- Réviser la nouvelle solution si besoin. Une fois l'adaptation faite, il faut tester la solution, par exemple sur une simulation ce qui peut en entraîner une révision.
- Retenir la solution. Une fois que la nouvelle solution est validée, un nouveau cas est créé et stocké dans la base de connaissances

Trois problèmes se dégagent lors de l'application d'un raisonnement par cas : la représentation des cas, la recherche de cas dans la base de connaissance et la création d'une fonction d'adaptation de solution. Les deux premiers sont des problèmes d'indexation, de filtrage et de sélection qui ne concernent pas vraiment notre sujet. Il est par contre intéressant de s'étendre un peu sur l'adaptation d'une solution.

Les deux principales approches se nomment adaptation transformationnelle et adaptation dérivative. La première réutilise directement les solutions passées en les transformant selon des lois dépendantes du domaine d'application et sans tenir compte de la manière dont ses solutions ont été trouvées. L'adaptation dérivative se base quant à elle sur les annotations expliquant les étapes de raisonnement qui ont mené à la solution réutilisée. Cela se présente sous forme de sous-objectifs à résoudre. Elle va appliquer ces mêmes étapes au cas courant en privilégiant les chemins pris par les cas similaires résolus. Comme le cas courant est légèrement différent, de nouveaux sous-objectifs doivent être introduits ce qui mène à une solution différente.

Au fil du temps, la base s'enrichit de nouveaux cas et le système devient apte à gérer de plus en plus de problèmes, le système apprend. Leur mécanisme de révision rapproche les CBR des systèmes d'apprentissage par renforcement, cependant ils représentent plus une architecture générique qu'ils ne répondent aux problèmes classiques d'apprentissage comme

le dilemme exploration vs. exploitation.

Le raisonnement par cas est particulièrement adapté au domaine médical, où les diagnostics et les prescriptions d'un médecin s'appuient naturellement sur son expérience. C'est d'ailleurs dans ce domaine où cette approche est la plus appliquée (Begum et al. 2011). Bien qu'il connaisse également certaines applications au problème du contrôle, le besoin d'une base de connaissance et d'une description exhaustive du problème en complique fortement l'instanciation à des systèmes du monde réel, à la dynamique complexe et parfois mal connue. Enfin, la principale critique faite au raisonnement par cas est qu'il se base sur peu de données pour inférer, ce qui peut conduire à des erreurs.

2.2.3.5 Bilan de l'apprentissage par renforcement

Il existe une multitude d'autres méthodes d'apprentissage par renforcement qui n'ont pas été abordées ici, notamment la recherche arborescente (Auer, Cesa-Bianchi et Fischer 2002) ou encore les techniques incluant des réseaux de neurones (Heinen et Engel 2010). Nous voyons cependant immédiatement que ce principe d'apprentissage est le plus intéressant pour notre problème. En effet, il propose des systèmes pensés pour s'adapter sur le long terme tout en étant en interaction constante avec leur environnement, ce qui correspond bien à la situation d'un système de contrôle.

Le principal défaut des méthodes existantes réside dans leur difficulté à être appliquées sur des problèmes réels hors simulations en laboratoire, trop complexes ou trop peu connus pour être modélisés. Une autre limite potentielle provient du grand nombre de cycles nécessaires avant d'aboutir à un comportement adéquat. Dans le cas du contrôle, cela peut mener à une dégradation irréversible du procédé avant d'être capable de le maîtriser.

2.2.4 Bilan de l'apprentissage artificiel

Ce paragraphe clôt le tour d'horizon de l'apprentissage artificiel. Les trois grandes familles d'apprentissage ont été présentées. Il est à noter que les techniques de chaque famille sont souvent combinées entre elles, et pas uniquement dans le cadre du méta-apprentissage ou de l'apprentissage semi-supervisé. Cela engendre une très grande quantité de variantes du même algorithme de base. Il en ressort une certaine souplesse mais aussi une difficulté certaine quand vient le moment de l'application à un problème concret.

Enfin, si l'approche la plus adaptée à notre problème est bien celle de l'apprentissage par renforcement, nous verrons dans la section suivante que des techniques supervisées et non supervisées ont été appliquées dans le cadre du contrôle de systèmes complexes.

2.3 Le contrôle intelligent

Dernier type de contrôle à être présenté dans ce document, le contrôle intelligent se concentre sur l'incorporation dans un contrôleur de méthodes issues de l'intelligence artificielle, et en particulier de l'apprentissage automatique. Les fondamentaux restent le plus

souvent les mêmes : nous retrouvons ainsi les PID d'une part et les classiques MIAC, MRAC et MPC du contrôle adaptatif d'autre part.

Les améliorations apportées par l'IA concernent généralement la mise à jour, l'amélioration ou même le remplacement par un mécanisme d'apprentissage du rôle joué par le modèle du procédé. Elles s'intéressent également au paramétrage des contrôleurs, notamment dans le cas des PID.

Cette section commence par un survol de deux méthodes d'IA utilisées dans le contrôle et qui ne sont pas de l'apprentissage : les sytèmes experts et la logique floue. Nous aborderons ensuite successivement des versions "intelligentes" des PID, MRAC, MIAC, MPC et contrôle dual. Enfin, nous terminons la présentation du contrôle intelligent avec quelques techniques plus exotiques ne reposant pas sur un algorithme ou une architecture classique. Une très grande partie des combinaisons possibles et imaginables ayant déjà été imaginée et publiée, il serait irréaliste de viser l'exhaustivité. Cependant l'échantillon présenté se veut caractéristique des méthodes et représentatif de leur diversité.

2.3.1 Des techniques d'IA utiles pour le contrôle

Si la grande majorité des apports de l'IA au contrôle concerne les mécanismes d'apprentissage, il en est deux qui n'en sont pas mais demeurent assez largement répandus. Les systèmes experts sont le couplage d'une base de connaissances et d'un moteur d'inférence tandis que la logique floue permet de raisonner dans l'incertain.

2.3.1.1 Systèmes experts

Un système expert est un programme interactif visant à reproduire le raisonnement et les connaissances d'un expert concernant un domaine particulier. Il est donc capable de répondre à des questions à partir de faits et de règles connus à l'avance. Il se compose d'une base de connaissances stockant des faits et des règles, alimentée au préalable par un expert du domaine, et d'un moteur d'inférence capable d'exploiter cette base.

Un système expert peut par exemple se situer sur une variable isolée, à la place d'un contrôleur PID (Konstantinov, Aarts et Yoshida 1993). Il peut aussi jouer un rôle de supervision interactive, signalant à un contrôleur humain pannes ou comportements anormaux, utilisant le moteur d'inférence pour décider quelle action entreprendre. Cependant, construire la base de connaissances adaptée à un système précis demande de le connaître parfaitement, ce qui est finalement le cas de peu de systèmes réels pour lesquels le recours à l'IA se fait sentir. En outre, il est impossible de gérer ainsi de potentielles situations inconnues rencontrées par le système. Seules des méthodes d'apprentissage telles que celles présentées dans la section 2.2 permettent de contourner ces limitations (Liao 2005).

2.3.1.2 Logique floue

Formalisée en 1965, la logique floue (*fuzzy logic*) est une extension de la logique classique permettant de prendre en compte les incertitudes et l'imprécision dans les raisonnements

(Zadeh 1988). Elle repose sur la théorie des sous-ensembles flous (*fuzzy sets*) qui redéfinit les ensembles classiques au moyen d'une fonction d'appartenance non plus binaire mais continue sur l'intervalle $[0;1]$ (Zadeh 1965).

En effet, appliquée à un élément x de l'ensemble E la fonction d'appartenance associée au sous-ensemble A, notée μ_A, vaut usuellement 1 si x appartient à A et 0 sinon. Dans la théorie des sous-ensembles flous, cette fonction représente un degré d'appartenance et prend une valeur réelle entre 0 et 1 selon que l'on soit plus ou moins sûr de l'appartenance de x à A (0 et 1 étant les valeurs pour lesquelles la certitude est établie). Cela amène la redéfinition des opérations sur les ensembles (union, intersection, etc) et de nouvelles notions comme par exemple :

- le *noyau* de A, c'est-à-dire les éléments dont le degré d'appartenance à A vaut 1.
- le *support* de A, c'est-à-dire les éléments dont le degré d'appartenance à A est strictement supérieur à 0.

Sur les mêmes principes, la logique floue autorise la valeur de vérité à parcourir un intervalle entre les valeurs faux (zéro) et vrai (un). Ainsi, chaque énoncé en logique floue n'est pas soit vrai, soit faux, mais possède un degré d'appartenance au sous-ensemble des énoncés vrais donné par la fonction d'appartenance μ. La conjonction, l'adjonction et la négation sont alors définies comme suit :

- le degré de vérité de la conjonction de deux énoncés de degré respectivement x et y est le plus faible des deux. Il faut donc choisir un opérateur $\top$ tel que $\top(x,y) \leq \min(x,y)$ pour faire la correspondance avec le "ET" de la logique classique. $\top$ est appelée une *t-norme*.
- le degré de vérité de l'adjonction de deux énoncés de degré respectivement x et y est le plus élevé des deux. Il faut donc choisir un opérateur $\perp$ tel que $\perp(x,y) \geq \max(x,y)$ pour faire la correspondance avec le "OU" de la logique classique. $\perp$ est appelée une *s-norme*.
- le degré de vérité de la négation d'un énoncé est son complément à 1.

La fonction d'appartenance ainsi que les opérateurs $\top$ et $\perp$, voire celui de la négation, sont à choisir selon la nature du problème. La fonction d'appartenance peut être linéaire, hyperbolique, exponentielle, ou de toute autre nature. Les opérateurs doivent quant à eux être choisis avec soin, selon les propriétés de l'algèbre booléenne que l'ont veut voir conservées ou non.

Il est possible de définir des règles IF-THEN-ELSE en logique floue afin d'implémenter un raisonnement. Les règles se déclenchent dès que le degré de vérité de leur condition est supérieur à zéro. Il faut ensuite une étape de *defuzzification* afin de prendre une décision parmi toutes les règles activées à plus ou moins fort degré. Classiquement, cette étape est un simple calcul de centre de gravité.

La logique floue est très utilisée en intelligence artificielle, notamment combinée de diverses manières avec les réseaux connexionnistes (*neuro-fuzzy*). On la retrouve également dans le domaine du contrôle de systèmes où l'on parle de contrôleurs flous. Les prochains paragraphes présentent quelques versions des techniques classiques de contrôle agrémentées

de logique floue et de mécanismes d'apprentissage.

2.3.2 Un exemple de PID intelligent

Nous avons vu dans la section 2.1.1 qu'un des points faibles des PID était leur difficulté à être instanciés, c'est donc naturellement que les travaux incluant de l'IA se sont penchés sur ce problème. Dans la méthode que nous présentons ici, la logique floue est utilisée pour améliorer la robustesse d'un PID fractionnaire tandis qu'un algorithme génétique aide le réglage de ses paramètres (Jesus et Barbosa 2013).

Les contrôleurs PID fractionnaires sont une variante des PID classiques dans laquelle au moins un des termes I ou D est fractionnaire (c'est-à-dire n'est pas une dérivée ou une intégrale d'ordre entier). Cela permet aux PID d'être plus robustes au retard. L'équation générale d'un PID fractionnaire est alors de la forme :

$$u(t) = K_p e(t) + K_i D_t^{-\alpha} e(t) + K_d D_t^{\beta} e(t)$$

où $e(t)$ est l'erreur, K_p, K_i et K_d sont les gains (respectivement proportionnel, intégral et dérivé), $D_t^{-\alpha}$ est l'intégrale fractionnaire d'ordre α et D_t^{β} la dérivée fractionnaire d'ordre β. Dans la pratique, $D_t^{-\alpha}$ et D_t^{β} sont approximées car trop coûteuses à calculer.

L'implémentation de Jesus et Barbosa 2013 intègre à un contrôleur PD^{β}+I (un PID dont le terme D est fractionnaire d'ordre β) une base de règles de logique floue afin de combiner les termes P et D. Son équation s'exprime donc de la manière suivante :

$$u(t) = [f(K_p e(t) + K_c D^{\beta} e(t)) + K_i \int e(\tau) d\tau] K_u$$

où f est la fonction réalisée par la base de règles floues et K_p, K_c, K_i et K_u les différents gains à régler. La figure 2.15 schématise ce contrôleur.

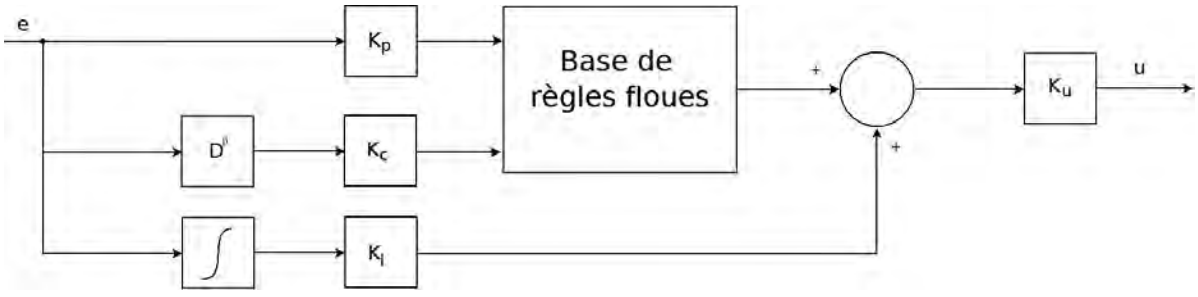

Figure 2.15 – Schéma d'un contrôleur PD^{β}+I flou.

Il reste maintenant le classique problème du réglage des différents gains. Ici, un algorithme génétique est appliqué pour en définir les valeurs optimales. Un chromosome comprend quatre gènes (un pour chaque gain) et la fonction d'évaluation est la mesure de l'erreur absolue intégrée sur le temps.

Les contrôleurs ainsi générés et paramétrés sont plus robustes et précis que les PID classiques et sont paramétrés automatiquement. Cependant, l'algorithme génétique pose un

problème en terme d'application. Il faut, en effet, exécuter de nombreux tests d'évaluation sur les populations générées. Réaliser ces tests sur le système réel peut être très coûteux, voire dangereux. Il faudrait alors disposer d'une simulation très fine du procédé à contrôler, ce qui est plutôt rare. En outre, si une simulation est possible, d'autres stratégies de contrôle à base de modèles seraient probablement plus appropriées. Enfin, un tel contrôleur est dépassé si le procédé évolue au cours du temps au point de rendre nécessaire un ajustement des paramètres.

2.3.3 Un exemple de MRAC intelligent

Nous avons vu que le défaut du contrôle adaptatif, dont les contrôleurs à modèle de référence (MRAC), était la nécessité de disposer d'un modèle. Outre la difficulté due à l'obtention d'un modèle adéquat pour le contrôle, il se pose le problème de la robustesse aux pannes. En effet, une avarie sur le système contrôlé va modifier son comportement et donc nécessiter une gestion particulière du point de vue du contrôle.

La résistance aux pannes est un problème à part entière et un sujet actif de recherche (Noura et al. 2009). Une solution courante dans les MRAC pour détecter et prendre en compte une panne telle que la perte d'efficacité, voire le blocage d'un effecteur repose sur une décomposition paramétrique du modèle : les termes non linéaires sont reformulés comme une combinaison linéaire de fonctions connues multipliées par un vecteur de paramètres inconnus qui sont à estimer. Or, cette décomposition n'est pas toujours faisable dans des cas concrets et mène à des calculs trop coûteux.

Fan et Song 2012 intègrent un réseau de neurones dans une architecture MRAC afin qu'il apprenne les bornes des perturbations dues aux pannes et les compense. Le réseau de neurones utilisé est un réseau RBF (pour *Radial Basis Function*), qui est un type particulier de perceptron à une couche cachée utilisant des fonctions à base radiale comme fonctions d'activation. L'équation du contrôle devient :

$$u(t) = -K_0 e(t) + \hat{K}(t)$$

où K_0 est calculé à partir du modèle, $e(t)$ est l'erreur courante et $\hat{K}$ dépend de la sortie du réseau de neurones. Les lourds calculs de linéarisation sont ainsi évités.

Cette méthode permet de gérer efficacement les pannes et les imprécisions (aussi bien du modèle que des capteurs) mais ne dispense pas d'avoir à construire un modèle du procédé.

2.3.4 Un exemple de MIAC intelligent

La tâche la plus cruciale dans un système de contrôle à identification de modèle (MIAC) est précisément l'identification d'un modèle du procédé contrôlé. Cette identification peut se faire hors ligne à partir de données expérimentales ou bien à l'aide de mesures en ligne. Habituellement, une structure de modèle est donnée au préalable et seule la valeur des paramètres est à trouver de manière à minimiser l'erreur entre la sortie du modèle et les données réelles.

Afin d'éviter l'utilisation d'une structure propre au procédé contrôlé, les réseaux de neurones ont été massivement utilisés en lieu et place de modèles (Nørgaard et al. 2000), en particulier les perceptrons multi-couches. Barreto et Araujo 2004 proposent une solution mettant en œuvre les cartes de Kohonen. Dans le cas d'un système non-linéaire SISO (*simple input, simple output*), le problème s'exprime en temps discret comme l'approximation de la fonction f telle que :

$$y(t+1) = f[y(t), ..., y(t-n_y+1); u(t), ..., u(t-n_u+1)] \tag{2.8}$$

où $u(t)$ et $y(t)$ sont la valeur de, respectivement, l'entrée et la sortie au temps t, et n_y et n_u le nombre d'échantillons mémorisés pour chacune d'entre elles. Cette fonction représente un procédé dont l'état suivant de sa sortie dépend des états précédents de son entrée et de sa sortie. Pour que le réseau soit capable d'apprendre le comportement d'un tel procédé, la dimension de l'entrée des neurones et du poids de leur connexion est passée de 1 à 2. ainsi, l'entrée d'un neurone $x(t)$ et le poids d'une connexion au neurone i, $w_i(t)$ sont notés :

$$x(t) = \begin{pmatrix} x^{in}(t) \\ x^{out}(t) \end{pmatrix} \text{ et } w_i(t) = \begin{pmatrix} \omega_i^{in}(t) \\ \omega_i^{out}(t) \end{pmatrix}$$

où $x^{in}(t)$ représente les arguments de la fonction f de l'équation 2.8 et $x^{out}(t)$ correspond à la valeur de sortie désirée, c'est-à-dire au terme $y(t+1)$ de la même équation. Ces informations sont mémorisées dans le poids des connexions, respectivement dans $\omega_i^{in}(t)$ et $\omega_i^{out}(t)$, par le biais de leur ajustement.

Lors de l'apprentissage, le neurone i le plus proche de l'échantillon d'entrée est sélectionné en se basant uniquement sur $x^{in}(t)$ et $\omega_i^{in}(t)$, mais l'ajustement se fait quant à lui en utilisant les deux dimensions. Ainsi, à la fin de l'apprentissage la partie $\omega^{out}(t)$ du poids de la connexion du neurone sélectionné (celui dont l'état $x^{in}(t)$ est le plus proche de l'état observé du procédé) est une approximation de la sortie du procédé $y(t+1)$.

La méthode peut s'étendre aux systèmes MIMO, la rendant ainsi utilisable dans de plus nombreux cas. Elle permet de se passer de la construction d'un modèle mais reste complexe à appliquer sur un cas concret. Le nombre de neurones et leur topologie initiale ainsi que de nombreux autres paramètres ne sont pas évidents à définir.

2.3.5 Un exemple de commande prédictive intelligente

En commande prédictive (MPC), le contrôle est basé sur les prévisions d'un modèle du procédé. Les meilleures actions possibles sont déduites de ces prévisions à l'aide d'algorithmes d'optimisation qui peuvent devenir très lourds en termes de temps de calcul, notamment si le modèle est de trop grand ordre. Une manière de simplifier un modèle tout en gardant les informations importantes qu'il contient est l'analyse en composantes principales (ACP), issue de l'apprentissage non supervisé.

Cette méthode a, par exemple, été appliquée sur des modèles de dynamique des fluides (Astrid et al. 2002). Comme la grande majorité des modèles mathématiques de procédés,

ils sont exprimés sous forme d'équations différentielles qui sont habituellement résolues numériquement aux prix de lourds calculs. Avant de pouvoir appliquer l'ACP, le modèle doit être discrétisé. Ainsi transformé, ses équations font apparaître des matrices plutôt que des dérivées partielles. Ces matrices sont d'autant plus grande dimension que l'ordre du modèle différentiel est important. En reformulant les matrices selon un critère heuristique propre à la méthode, l'ACP réduit leur dimension et permet ainsi de rester dans une limite acceptable de temps de calcul. Le modèle réduit est alors exploitable dans un algorithme classique de commande prédictive (voir section 2.1.2.3).

Il faut cependant bien prendre garde au paramétrage de l'ACP, notamment du critère heuristique, car il a une forte influence sur la précision du modèle obtenu, mais aussi sur l'ampleur de la réduction. En outre, l'ACP ne se comporte pas très bien en cas de forte non linéarité.

2.3.6 Un exemple de contrôle dual intelligent

La théorie du contrôle dual cherche à apprendre le comportement du procédé contrôlé en procédant à la fois à des actions-sondes visant à améliorer les connaissances et à des actions de contrôle proprement dites. Le problème est alors de trouver le bon compromis entre les deux types d'action. Trop d'actions-sondes est dangereux car le système devient instable et il est possible de le détériorer. Mais trop d'actions de contrôle mène à un contrôle lent et sous-optimal, car l'incertitude sur le procédé reste grande et donc les contrôles de faible amplitude. Résoudre ce compromis revient à résoudre la difficile équation de Bellman (équation 2.2). C'est là qu'intervient l'apprentissage automatique, et notamment les réseaux de neurones.

En effet, il est possible d'utiliser un ou plusieurs perceptrons afin d'en approximer la solution dans certains cas. Par exemple Fabri et Bugeja 2013 s'intéressent aux systèmes MIMO non linéaires mais affines en contrôle, de la forme :

$$y_k = f(x_{k-1}) + G(x_{k-1})u_{k-1} + \epsilon_k$$

où $y_k \in \mathbb{R}^s$ est le vecteur des s sorties, $u_k \in \mathbb{R}^s$ le vecteur des s entrées et x_{k-1} le vecteur d'état du système (comprenant l'historique des entrées et sorties). f et G sont respectivement un champ vectoriel et une matrice inconnus, représentant la dynamique non-linéaire du système tandis que ϵ_k est un vecteur modélisant le bruit. Deux perceptrons sont utilisés, le réseau $\hat{f}$ est chargé d'apprendre f et le réseau $\hat{g}$ d'apprendre G. Chacun des deux réseaux possède une couche cachée et utilise une fonction d'activation sigmoïde (équation 2.6). L'activité des réseaux dépend du poids de leurs connexions.

L'inconvénient du mécanisme d'apprentissage classique des perceptrons, la rétropropagation de l'erreur (voir section 2.2.1.5), est qu'il n'est pas du tout adapté au temps réel nécessaire pour le contrôle dual. La solution proposée ici est d'utiliser les filtres de Kalman. Les filtres de Kalman sont des estimateurs récursifs de paramètres, c'est-à-dire qu'ils se basent sur les mesures courantes (possiblement bruitées et incomplètes) et sur l'estimation de l'état

précédent pour estimer l'état courant (Kalman et Bucy 1961). Ils se décomposent en deux phases :

- La première phase, dite de prédiction, estime l'état et la covariance du paramètre considéré en se basant sur l'état précédent.
- La phase de mise à jour améliore les précédentes estimations en prenant en compte les mesures actuelles.

L'algorithme de base est optimal dans les cas linéaires. Des variantes comme les filtres de Kalman étendus, qui utilisent les séries de Taylor pour linéariser, ou les filtres de Kalman inodores (de l'anglais *unscented Kalman filters*), qui appliquent une transformation particulière aux données, permettent d'être efficace sur des systèmes non linéaires. Ce sont ces variantes qui sont étudiées dans la méthode présentée ici.

On représente les poids des deux réseaux dans un même vecteur z_k, et on note le vecteur des poids optimaux z_k^*. On peut alors reformuler l'équation du système contrôlé comme :

$$y_k = h(x_{k-1}, u_{k-1}, z_k^*) + \epsilon_k$$

La sortie du système contrôlé est donnée par la fonction non linéaire inconnue h qui dépend des poids optimaux z^*. L'observation de la sortie apporte ainsi au filtre utilisé une indication sur la qualité de l'estimation courante.

Les contrôles appliqués sont ensuite déterminés par une loi qui prend en compte :

- les estimations de f et G données par les réseaux de neurones
- l'estimation de l'incertitude donnée par le filtre de Kalman utilisé
- l'état courant des sorties
- des matrices à définir par l'utilisateur : Q_1, Q_2 et Q_3

Ces matrices permettent de pondérer l'attention portée aux incertitudes. Si les incertitudes sont prépondérantes, on se retrouve dans le cas d'un contrôle précautionneux car le contrôleur ne va faire que des actions de faible amplitude. Si les incertitudes sont ignorées, aucune action sonde n'est entreprise puisque le contrôleur est sûr de lui, ce qui peut engendrer une instabilité du système.

Cette méthode permet, grâce aux réseaux de neurones et aux filtres de Kalman, de réduire la résolution complexe d'une équation au paramétrage de quelques matrices. Cependant, le compromis délicat est au bout du compte à réaliser par l'utilisateur humain qui va paramétrer le contrôleur. Enfin, l'étude se limite à une première convergence du contrôle. Le contrôleur peut avoir du mal à se réadapter si le procédé contrôlé évolue fortement au cours du temps.

2.3.7 Autres exemples

Les techniques de contrôle intelligent présentées jusqu'à présent étaient des classiques du contrôle agrémentées d'intelligence artificielle. Mais l'IA a également apporté de nombreuses approches inédites qui ne reposent pas directement sur celles de la section 2.1. Nous en présentons deux d'entre elles. La première est basée sur les systèmes de classeurs, tandis que la seconde combine quatre techniques d'IA différentes.

2.3.7.1 Contrôle distribué à base de systèmes de classeurs

Par leur système d'appariement et de sélection d'action sur l'environnement, les systèmes de classeurs (LCS) sont particulièrement adaptés au problème du contrôle. Ils peuvent même s'y appliquer directement, et de manière distribuée (Bull et al. 2004).

On s'intéresse ici au cas simulé d'un réseau routier de quatre carrefours, chacun étant équipé d'un LCS pour contrôler l'ensemble des feux tricolores du carrefour. Nous avons donc quatre contrôleurs agissant à différents endroits d'un système plus large (l'ensemble du réseau routier simulé), il s'agit d'un contrôle distribué. Chaque LCS doit apprendre la durée des phases (autoriser la circulation dans un sens ou dans l'autre) permettant un trafic optimal. Pour cela, il est stimulé à chaque cycle par la perception de la longueur de la plus grande file d'attente du carrefour et d'un signal de renforcement calculé à partir de cette mesure.

Plusieurs tests ont été conduits afin de mesurer l'impact des différents paramètres comme le nombre de règles dans la base (entre 400 et 1600), le taux de mutation de l'algorithme génétique, ou encore le taux d'apprentissage pour le renforcement. Chaque carrefour est équipé d'une instance identique de LCS. Contrairement aux précédentes expériences n'impliquant qu'un seul LCS isolé, l'étude montre que ces paramètres n'ont pas ici d'influence déterminante sur les performances.

Cependant, d'autres choix d'instanciation demeurent cruciaux tels que le mécanisme de sélection d'action ou bien le choix du signal de renforcement. Dans le cas étudié, les LCS n'offrent une performance bénéfique uniquement si ces fonctions sont bien construites et paramétrées. Or il n'existe pas de règle permettant de les choisir a priori. On s'attend donc à ce que l'application de LCS sur des cas réels soit un problème difficile.

2.3.7.2 Contrôle distribué hybride

Une seconde approche, qualifiée par les auteurs d'hybride, fait appel à quatre techniques d'IA qu'elle combine dans un système de contrôle distribué : les réseaux de neurones, la logique floue, les algorithmes génétiques et l'apprentissage par renforcement (Choy, Srinivasan et Cheu 2006).

Comme la méthode précédente, elle s'intéresse au contrôle des feux de signalisation des carrefours d'un réseau routier. Le contrôleur de chaque carrefour utilise des règles de logique floue pour produire des plans de signaux (les durées de chaque phase des feux). Ces règles sont générées par un algorithme génétique et représentées dans un réseau de neurones dont les paramètres sont ajustés par renforcement. L'apprentissage du réseau de neurones se fait en temps réel, en trois grandes étapes :

- La première étape est une mise à jour des paramètres du réseau (par exemple le taux d'apprentissage de chaque neurone). Celle-ci se fait par renforcement.
- Les poids des connexions sont ensuite mis à jour selon une méthode semblable à celle des cartes de Kohonen.
- Enfin, le signal de renforcement est utilisé dans le calcul de la fonction d'évaluation d'un algorithme génétique. Si le résultat de l'évaluation est en dessous d'un seuil prédéfini,

l'algorithme génère de nouvelles règles (matérialisées par de nouvelles connexions dans le réseau).

Le signal de renforcement est calculé à partir d'estimations de l'état du trafic et de mesures, et est propagé dans chacun des contrôleurs locaux. Le réseau de neurone de chaque contrôleur représente de manière assez directe les règles de logique floue qu'il contient. Il est composé de cinq couches (figure 2.16) :

- Une couche *entrée* reçoit les valeurs mesurées sur le système contrôlé. Chaque neurone de cette couche correspond à une variable perçue.
- Une couche *fuzzification* décrit de manière floue les entrées. Chaque neurone de la couche précédente est connecté à 3 neurones de la couche fuzzification, qui correspondent à trois sous-ensembles flous quantifiant la valeur (faible, moyen et fort). Les poids des connexions entre ces deux couches jouent le rôle de fonction d'appartenance.
- Une couche *implication* applique un opérateur de t-norme sur la précédente pour représenter l'état du système de manière plus complète. Chaque neurone de cette couche représente donc une conjonction floue des valeurs des variables floues de la couche précédente.
- Une couche *conséquence* représente les différents plans possibles de signal. Elle applique un opérateur de s-norme sur la précédente. Un neurone-plan est ainsi activé si la disjonction floue de ses entrées le permet.
- Enfin, une couche *defuzzification*, composée d'un unique neurone, applique la méthode du centre de gravité pour agréger les différents plans activés et calculer la sortie finale (un plan de signal à appliquer au carrefour).

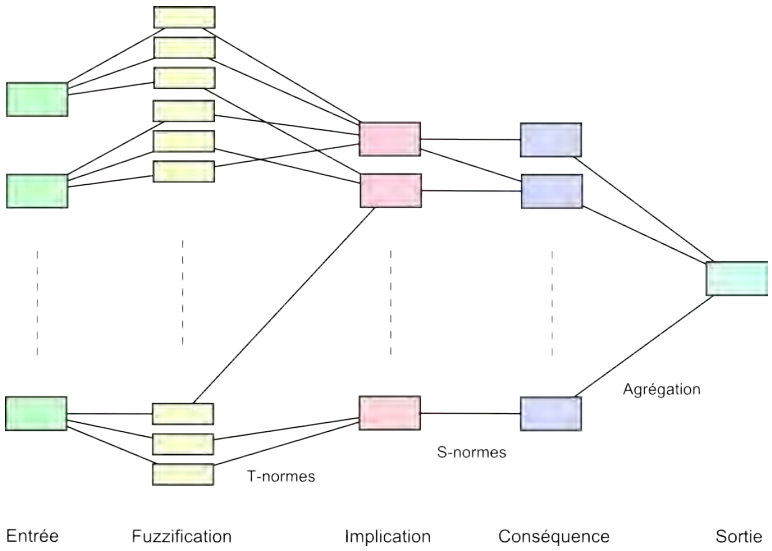

Figure 2.16 – Le réseau de neurones flou à cinq couches.

Les connexions entre les couches *fuzzification* et *implication* sont représentées dans les chromosomes utilisés par l'algorithme génétique. Celui-ci peut ainsi les modifier afin que le réseau maximise une fonction d'évaluation. Cette fonction est basée à la fois sur le signal de

renforcement et sur une mesure de satisfaction de la connexion. Le nombres d'individus par génération, le nombre d'individus sélectionnés pour le croisement, le taux de mutation et autres paramètres de l'algorithme sont à définir par l'utilisateur.

La distribution des contrôleurs et la combinaison judicieuse de techniques d'IA permet ici d'effectuer un apprentissage en temps réel du contrôle d'un système complexe de manière à optimiser un critère unique (la fluidité du trafic). Cependant si les paramètres du réseau de neurones sont appris, de nombreux autres (ceux de l'algorithme génétique, du signal de renforcement, l'initialisation du réseau, ainsi que d'autres propres à la méthode) restent à définir empiriquement. En simulation, il est relativement facile d'effectuer des tests afin de déterminer la meilleure valeur pour ces paramètres. Mais dans le cas d'un système réel cela est très souvent long et coûteux et pose donc un sérieux problème d'instanciation. En outre, pour la plupart des procédés à multiples entrées et sorties le contrôle implique de faire un compromis entre plusieurs critères parfois contradictoires, ce que ne permet pas cette méthode.

2.3.8 Bilan du contrôle intelligent

Ce document ne présente qu'une petite partie de la légion de variations et de combinaisons de méthodes existantes d'IA (à la fois entre elles et avec des approches de contrôle). Ont été laissées de côté des approches combinant notamment SVM et perceptrons (Suykens, Vandewalle et De Moor 2001), inférence d'arbres de décisions et algorithmes génétiques (Su et Shiue 2003), réseaux bayésiens et k plus proches voisins (Lazkano et al. 2007), ou encore contrôle adaptatif et réseaux de Hopfield (Wang et Hung 2010). Les outils les plus largement utilisés restent probablement les réseaux de neurones et la logique floue.

Dans cette immense variété de contrôleurs intelligents, chacun a ses propres avantages et inconvénients, si bien qu'il est très difficile d'en faire une généralisation pertinente. Certaines conjonctions intéressantes de critères semblent néanmoins ne pas être présentes. Par exemple, il n'existe pas, à ma connaissance, de contrôleur générique capable d'apprentissage en ligne et en temps réel, effectuant une optimisation multi-critères sur un système MIMO tout en étant très facilement instanciable et capable de passer à l'échelle en terme de nombre de critères d'optimisation et de paramètres contrôlés.

Globalement, on note une complexification des contrôleurs intelligents, qui combinent de plus en plus de techniques et les affinent. Chaque fois que des paramètres sont à estimer, on empile une nouvelle technique qui va s'en charger. Mais celle-ci apporte souvent son propre lot de paramètres à définir, et ainsi de suite. Cela semble en accord avec la loi de la variété requise puisque les procédés visés sont eux aussi de plus en plus complexes.

Le tableau 2.4 reprend les critères d'évaluation des méthodes de contrôle. La plupart des algorithmes utilisés ne dépendent pas directement du système contrôlé et sont donc assez génériques. Cela provient de leur approche "boîte noire", c'est-à-dire que leurs connaissances reposent sur des données acquises sur les entrées et les sorties du procédé contrôlé, et non sur un modèle décrivant son fonctionnement. Cette approche est bien sûr permise par leurs capacités d'apprentissage, qui peuvent concerner aussi bien la paramétrisation du contrôleur

que la loi de contrôle elle-même. En outre, un apprentissage en ligne et en temps réel confère généralement au contrôleur une bonne capacité d'adaptation. En revanche l'accent est rarement mis sur le travail d'instanciation à fournir pour appliquer de tels contrôleurs sur des cas réels, hors simulation. La complexification des contrôleurs entraîne une plus grande difficulté à leur application industrielle. Celle-ci demande en effet d'être à la fois expert du domaine d'application et fin connaisseur des méthodes utilisées par le contrôleur.

Table 2.4 – Bilan du contrôle intelligent.

Critère	Contrôle intelligent
Généricité	+ +
Instanciation	- -
Adaptativité	+ +
Apprentissage	Variable (de limité à très important)

C'est en partie pour cette raison que l'on retrouve assez peu d'applications concrètes dans le champ des moteurs à combustion. Nous en présentons quelques-unes dans la section suivante.

2.4 Les applications au contrôle de moteurs

Jusqu'à maintenant, nous avons présenté le problème du contrôle et les techniques qui l'abordent d'un point de vue général. Il est maintenant temps de s'intéresser plus particulièrement au cas concret du contrôle de moteurs à combustion dans l'industrie automobile. Comme nous l'avons vu dans le chapitre précédent, le contrôle d'un moteur se divise en deux couches :

- La première constitue l'asservissement du moteur (la couche basse). Il s'agit de s'assurer que les effecteurs appliquent correctement la consigne qui leur est envoyée.
- La deuxième est le contrôle à proprement parler. Elle transforme l'angle de la pédale d'accélération en demande de couple, puis en consignes pour la couche basse.

La première couche est généralement composée de PID et de contrôleurs MPC. La deuxième couche est quant à elle assurée par un ensemble de stratégies de commande reposant sur des modèles physiques. Ces stratégies sont composées de blocs fonctionnels interdépendants (circuit d'air, avance à l'allumage, etc) et sont, le plus souvent, conçues manuellement par les ingénieurs, bien que quelques blocs fonctionnels puissent faire appel à des MPC (Del Re et al. 2010). La difficulté est alors d'optimiser les paramètres de ces stratégies, autrement dit, de calibrer le moteur.

Aussi, cette section commence par présenter quelques méthodes de contrôle avant d'introduire des approches de calibration automatique.

2.4.1 Méthodes de contrôle

Condition préalable à l'utilisation d'un algorithme de commande prédictive, l'obtention d'un modèle repose sur la connaissance des processus physiques mis en jeu dans le moteur. Depuis les années 60, divers modèles de moteurs ont été proposés, comme par exemple le modèle de Borman pour les moteurs diesels (Borman 1964). Nous présentons ici le modèle de Jankovic, qui est le plus utilisé aujourd'hui pour les moteurs turbo-diesels. Nous aborderons ensuite des techniques de contrôle, dont certaines qui utilisent ce modèle.

2.4.1.1 Modèle de Jankovic

Le modèle de Jankovic exprime les variations de pression, de masse de gaz et de proportions de gaz brûlés à l'intérieur des collecteurs d'admission et d'échappement à partir des caractéristiques physiques du moteur et des différents débits contrôlés (Jankovic et Kolmanovsky 2000). Il existe une première version de ce modèle, dite d'ordre complet :

$$
\begin{aligned}
\dot{m_1} &= W_c + W_{egr} - W_e \\
\dot{m_2} &= W_e - W_{egr} - W_t + W_f \\
\dot{p_1} &= \frac{\gamma R}{V_1}(W_c T_c + W_{egr} T_{egr} - W_e T_1) \\
\dot{p_2} &= \frac{\gamma R}{V_2}((W_e + W_f) T_e - W_{egr} T_2 - W_t T_2) \\
\dot{F_1} &= \frac{W_{egr}(F_2 - F_1) - W_c F_1}{m_1} \\
\dot{F_2} &= \frac{W_e[15.6(1 - F_1) + (AF + 1)F_1]/(AF - 1) - W_e F_2}{m_1}
\end{aligned}
$$

où m_1 et m_2, p_1 et p_2, F_1 et F_2 sont respectivement les masses de gaz, la pression et la proportion de gaz brûlés dans les collecteurs d'admission et d'échappement. V_1 et V_2, et T_1 et T_2 sont le volume et la température de chacun des collecteurs. T_c est la température du compresseur, T_e la température à l'échappement et AF le ratio air/carburant. Issues de la thermodynamique, γ et R sont respectivement la capacité thermique spécifique du système et la constante spécifique des gaz. Enfin, W_c, W_{egr}, W_e, W_t et W_f sont les débits massiques, respectivement dans le compresseur, dans la vanne EGR, dans le moteur entier, dans la turbine et de carburant. Ils sont pour la plupart manipulables à l'aide d'effecteurs et constituent donc les entrées du moteur pour ce modèle.

Cependant, trop lourde du fait de ses trop nombreux paramètres et de sa complexité calculatoire, cette version n'est jamais utilisée. En effet, il faut en estimer correctement la valeur de ses paramètres afin que le modèle corresponde parfaitement à l'instance de moteur contrôlée et à l'utilisation qui en est faite, ce qui est un problème très compliqué lorsque ces paramètres sont nombreux. En outre, elle demande des mesures très difficiles à obtenir sur un moteur réel (par exemple la proportion de gaz brulée dans chacun des collecteurs). Aussi cette version est généralement laissée de côté. On lui préfère une plus simple, ne comprenant

que trois paramètres et s'intéressant uniquement aux variations de pression. Cette version, appelée *modèle d'ordre 3*, s'exprime ainsi :

$$\dot{p_1} = k_1(W_c + u_1 - k_e p_1) + \frac{\dot{T_1}}{T_1} p_1$$
$$\dot{p_2} = k_2(k_e p_1 - u_1 - u_2) + \frac{\dot{T_2}}{T_2} p_2$$
$$\dot{P_c} = \frac{1}{\tau}(\eta_m P_t - P_c)$$

où P_c et P_t sont la puissance du compresseur et celle de la turbine, η_m est le rendement mécanique du turbo, u_1 et u_2 sont les variables contrôlées (en fait les débits EGR et de la turbine, $u_1 = W_{egr}$ et $u_2 = W_t$). Les trois coefficients k_1, k_2 et k_e sont les paramètres du modèle à définir par l'utilisateur. Pour simplifier un peu plus le modèle, les termes $\frac{\dot{T_1}}{T_1} p_1$ et $\frac{\dot{T_2}}{T_2} p_2$ sont souvent ignorés. En effet, ils sont proches de zéro, la température variant peu dans les plages de fonctionnement considérées par la plupart des méthodes de contrôle.

Le nombre de paramètres est réduit, mais leur estimation demeure un problème délicat. Divers algorithmes de régression peuvent être utilisés comme le classique algorithme de Gauss-Newton (GNA, Wang 2012) ou encore l'algorithme de Levenberg-Marquardt (LMA, Moré 1978) mais sont difficiles à mettre en œuvre avec un vrai moteur.

Pour être optimale, la valeur des paramètres devrait changer selon la plage de fonctionnement du moteur considérée. En effet, le régime, la charge, ou encore le débit de carburant impactent directement le comportement du moteur et ainsi la conformité du paramétrage du modèle. En outre, il faudrait réajuster ces paramètres au cours de la vie du moteur pour en suivre l'usure et l'encrassement. Néanmoins, cela n'empêche pas ce modèle d'être utilisé pour le contrôle d'un moteur dans des situations bien définies.

2.4.1.2 Contrôle prédictif appliqué aux moteurs

Comme dans tout contrôleur prédictif basé sur un modèle, la principale difficulté est de gérer la non linéarité du modèle. En effet, celle-ci rend très complexe le problème d'optimisation visant à calculer les actions à effectuer. Diverses solutions ont été entreprises.

Par exemple, la commande prédictive généralisée non linéaire à temps continu (NCGPC) modifie le vecteur de sortie du modèle de Jankovic en choisissant d'autres variables à observer. Ce changement est principalement motivé par l'impossibilité de mesurer les variables initialement impliquées. La loi de commande est ensuite calculée en s'appuyant sur une matrice de découplage, sur une extension bien choisie du vecteur d'états du modèle (afin de rendre la matrice inversible) et sur les séries de Taylor (Dabo et al. 2008).

Une autre approche combine des modèles linéaires locaux (relativement à l'espace d'états du moteur), et passe de l'un à l'autre à la volée en fonction de l'état courant du moteur pour y appliquer une commande prédictive classique (Ferreau 2006).

Ces méthodes sont efficaces et capables d'être exécutées par un calculateur embarqué. Cependant, elles se limitent à certaines fonctions du moteur (par exemple la vanne EGR

pour la NCGPC), le contrôle global étant réalisé par un assemblage *ad hoc* de telles méthodes. Elles ne sont possibles que parce qu'il existe un modèle du moteur qu'elles considèrent, et applicables à la condition que celui-ci soit bien paramétré.

D'autres applications incluent des réseaux de neurones (Ouladsine, Bloch et Dovifaaz 2005), ou encore de la logique floue (Nikzadfar, Noorpoor et Shamekhi 2012) mais restent cantonnées à un contrôle local, d'un seul actionneur (un seul bloc fonctionnel) et sujettes à une paramétrisation importante.

2.4.2 Auto-calibration

La calibration d'un moteur peut comprendre celle des éventuels modèles utilisés, mais consiste surtout à la paramétrisation du contrôle lui-même. Le but est d'optimiser le fonctionnement du moteur en réglant de manière fine les consignes envoyées aux contrôleurs de la couche basse. Les méthodes classiques, très largement utilisées dans l'industrie, produisent des cartographies statiques reliant les états du moteurs aux consignes pour chaque effecteur. Elles ont le défaut de ne pas être optimales pour les régimes transitoires (Atkinson et Mott 2005) mais sont surtout très longues (et donc coûteuses) à mettre en œuvre, impliquant de nombreuses phases de tests et de mise au point, ainsi que l'utilisation de modèles. Aussi, le besoin de techniques de calibration automatique est très présent. Les termes d'auto-calibration et de calibration automatique sont employés indifféremment dans la littérature pour désigner aussi bien l'exécution automatique de tests que l'optimisation automatique des réglages du contrôleur. C'est bien sûr de cette dernière dont il est question ici.

Les recherches les plus avancées se concentrent sur l'obtention d'un contrôleur capable de s'auto-calibrer pendant que le moteur est en fonctionnement, en apprenant en temps réel les réglages optimaux. Basé sur une modélisation du moteur comme un processus de décision markovien (MDP), un premier algorithme appelé POSCA a été développé. Celui-ci est capable d'apprendre en ligne les meilleures actions à appliquer grâce à un signal de renforcement. Cependant, la complexité combinatoire du problème n'autorise l'optimisation que d'une unique variable. La solution pour passer à de plus grandes dimensions se situe dans la décentralisation (Malikopoulos, Assanis et Papalambros 2009). Toujours en se basant sur une modélisation MDP du moteur, les auteurs ont donc conçu un contrôleur distribué.

Plutôt qu'un seul contrôleur central décidant de toutes les actions sur toutes les variables, chaque variable se voit attribuer un contrôleur local. Ces contrôleurs sont placés aléatoirement au sein d'une hiérarchie. Lors de l'apprentissage, chaque contrôleur choisi une action au hasard parmi l'ensemble des actions possibles. Le premier contrôleur de la hiérarchie le fait avec une probabilité uniforme conditionné par l'état courant du moteur :

$$p(a_k|s_k) = \frac{1}{|A|}, \forall a_k \in A, \forall s_k \in S$$

où A est l'ensemble des actions possibles pour le premier contrôleur et S l'ensemble des états du moteur. L'astuce est que le second contrôleur fait de même, mais la probabilité est cette-fois-ci conditionnée par l'action sélectionnée par le premier contrôleur, et non par l'état

du moteur :

$$p(\beta_k|\alpha_k) = \frac{1}{|B|}, \forall \beta_k \in B, \forall \alpha_k \in A$$

où B est l'ensemble des actions possibles pour le second contrôleur. Il en va de même pour le troisième contrôleur, et ainsi de suite. En plus d'un changement d'état du moteur, chaque action provoque un signal de renforcement (qui découle de la modélisation du moteur en MDP). Après un certain temps, chaque contrôleur local a exploré l'ensemble de ses actions possibles. POSCA est alors utilisé en parallèle par chacun d'entre eux pour calculer la meilleure stratégie de contrôle, c'est-à-dire la distribution de probabilités (toujours conditionnée de la même manière) qui maximisera le signal de renforcement.

La méthode a été testée avec succès pour le contrôle de deux variables (la durée de l'injection, et la position des ailettes du turbo), d'un moteur diesel simulé avec des critères de minimisation sur la consommation de carburant et l'émission de gaz polluant (qui ne sont pas contradictoires).

Malgré le frein de la modélisation en processus de décision markovien, qui peut être un lourd travail à faire pour chaque couple critères d'optimisation et moteur, cette approche montre bien l'intérêt de la distribution du contrôle. Cependant, aucun calcul visant à améliorer les actions n'est effectué avant l'exploration complète des possibilités, cela afin de garantir l'optimalité du résultat. Or, cette phase d'exploration peut être très longue si plus de deux variables sont en jeu. L'apprentissage n'est ici pas réellement distribué et effectué entièrement en amont de la décision des meilleures actions à entreprendre. Aussi, si le moteur ou les critères d'optimisation évoluent, il faudra recommencer la phase d'exploration (tout en ajustant la modélisation MDP).

2.4.3 Bilan du contrôle de moteurs

Actuellement, l'approche la plus répandue du contrôle de moteurs est la commande prédictive basée sur les modèles. Elle peut s'utiliser dans les deux couches du contrôle et s'appuyer sur des modèles solides développés depuis une cinquantaine d'années. En outre, la difficulté d'appliquer les méthodes de contrôle intelligent freine leur installation dans ce domaine où les contraintes techniques sont fortes. Par exemple, l'indisponibilité de certaines mesures lors du fonctionnement du moteur prive les contrôleurs d'une partie de l'information qui pourrait leur être nécessaire.

Un enjeu important pour l'industrie est la rapidité de mise au point d'un contrôleur. Chaque nouveau moteur s'accompagne en effet d'un long travail pour lui fournir un contrôleur adapté. La calibration prend une part importante de ce travail. Étant donné la structure en deux couches des contrôleurs, elle revient à faire la correspondance entre la consigne, l'état du moteur et les actions à entreprendre. Cela équivaut en fait à contrôler le moteur asservi.

Aussi, l'apprentissage (aussi bien du contrôle que de la calibration) devient crucial, il permet de gagner du temps à l'instanciation et d'obtenir de meilleurs résultats. Il permet aussi de se passer de modèle, ce qui est un avantage pour accélérer la prise en compte de nouvelles technologies moteur. Il est cependant souvent figé : une fois réalisé, les résultats ne

sont plus mis en cause. Or un moteur s'use avec le temps et le contrôle doit en tenir compte. Le prochain défi est donc l'obtention d'un contrôleur capable de s'adapter, sur la durée, aux changements du moteur.

2.5 Conclusion

Ce chapitre a présenté les trois grandes familles de systèmes de contrôle que sont les PID, le contrôle adaptatif et le contrôle intelligent. Le tableau 2.5 en récapitule les caractéristiques principales. L'accent a été mis sur l'importance des techniques d'intelligence artificielle (en particulier des algorithmes d'apprentissage) et la diversité de leur mise en œuvre dans les systèmes de contrôle.

TABLE 2.5 – Tableau récapitulatif du contrôle.

Critère	PID	Contrôle adaptatif	Contrôle intelligent
Généricité	+	+	+ +
Instanciation	- -	- -	- -
Adaptativité	- -	+	+ +
Apprentissage	Aucun (la connaissance du procédé est implicitement contenue dans le paramétrage	Limité (ajustement de paramètres d'une structure fixe)	Variable (de limité à très important)

Cet état de l'art souligne également certaines limites des méthodes actuelles. La principale est celle des lourdes tâches de spécification et de paramétrisation nécessaires à leur application sur un système particulier. Pour les algorithmes les plus anciens, ce travail d'instanciation consiste en l'apport de connaissances sur le système contrôlé, par exemple par le biais des paramètres d'un PID ou du modèle mathématique des contrôleurs adaptatifs. Les techniques d'apprentissage automatique permettent d'apprendre tout ou partie de ces connaissances. Mais, étant plus complexes, elles conservent un degré de paramétrisation important et ne font au final que déplacer le travail d'instanciation vers des problèmes propres aux algorithmes utilisés. Un contrôleur devrait être facile à instancier. L'utilisateur ne devrait pas avoir besoin de connaissances techniques sur l'algorithme employé, ni devoir fournir un lourd travail de paramétrisation. Un problème lié est la division du contrôle en plusieurs couches. Un même système de contrôle devrait être capable de transformer directement une consigne en action de bas niveau sur le procédé.

Une caractéristique importante que doit avoir un contrôleur est la capacité de s'adapter à l'évolution du système contrôlé. L'apprentissage ne doit pas être figé. La dynamique d'un système réel se modifie au cours du temps, à mesure qu'il s'use, et un contrôleur doit en tenir compte. Il doit également s'adapter lorsque des avaries surviennent sur les capteurs et modifient la perception qu'il a du procédé.

Enfin, si plusieurs techniques existantes de contrôle intelligent permettent de contrôler la non-linéarité de certains systèmes, peu sont dans le même temps capables de passer à

l'échelle. Contrôler plusieurs variables tout en conciliant plusieurs objectifs et contraintes reste un problème ouvert lorsqu'il s'agit de l'appliquer à un système réel.

Les besoins se situent donc sur trois axes :

- un système de contrôle facile à appliquer à une instance particulière de procédé
- un système de contrôle capable de suivre l'évolution du procédé, autrement dit capable d'apprendre en parallèle du contrôle
- un système de contrôle capable de passer à l'échelle, c'est-à-dire de gérer simultanément un grand nombre de variables contrôlées et de critères d'optimisation.

Le premier implique un système se passant de modèle prédéfini, et possédant des paramètres en petit nombre et faciles à déterminer. Le deuxième axe oriente la solution vers la classe des algorithmes d'apprentissage par renforcement, dont l'objet serait la fonction de contrôle elle-même et non un modèle ou sa calibration. L'apprentissage devrait en outre être suffisamment contrôlé pour ne pas risquer de dégrader le procédé. Enfin, le troisième axe encourage un contrôle distribué.

Parmi toutes les approches d'apprentissage et d'intelligence artificielle qui ont été abordées dans ce chapitre, il en est une qui a été volontairement omise et qui pourrait bien répondre à ces besoins : les systèmes multi-agents. Ils sont l'objet du chapitre suivant.

CHAPITRE 3

Systèmes multi-agents et coopération

L'objectif de ce chapitre est d'introduire les systèmes multi-agents (SMA) et de montrer en quoi cette approche est intéressante pour le contrôle de systèmes complexes. Les concepts principaux sont présentés avant d'aborder quelques techniques de contrôle et d'apprentissage utilisant les SMA. La suite du chapitre se focalise sur la notion d'auto-organisation et présente une vision particulière des SMA, fondée sur la coopération.

3.1 Les systèmes multi-agents

Les systèmes multi-agents (SMA) sont des systèmes composés de plusieurs entités autonomes en interaction que l'on appelle agents. La distribution au sein du système de la connaissance, des calculs, ou encore du contrôle en est la principale caractéristique. Ils appartiennent ainsi à la branche que l'on appelle distribuée de l'intelligence artificielle (IAD), celle qui s'intéresse à la résolution collective de problèmes, c'est-à-dire à l'élaboration de comportements au sein d'un collectif menant à la réalisation d'une tâche globale particulière.

Cette section commence par définir ce qu'est un agent, puis s'intéresse aux interactions entre plusieurs d'entre eux pour aborder le principe de système multi-agent. Quelques techniques multi-agents de contrôle et d'apprentissage sont ensuite présentées.

3.1.1 Qu'est-ce qu'un agent ?

Le concept d'agent est apparu avec les premiers programmes destinés à s'exécuter sur le long terme, au sein d'un environnement dans lequel ils jouissent d'une certaine autonomie d'action. Le terme se retrouve par exemple dans la littérature de l'apprentissage par renforcement.

Leur définition a depuis été enrichie et affinée, notamment par Wooldridge et Jennings 1995 et Ferber 1999. Un consensus s'est établi autour d'une liste de propriétés et d'un schéma d'exécution présentés ci-après. Un agent est une entité physique ou logicielle qui :

- est autonome,
- existe au sein d'un environnement qu'elle est capable de percevoir et sur lequel elle peut agir,
- possède une représentation partielle de cet environnement,
- est capable de communiquer avec d'autres agents,
- possède des ressources,
- possède des compétences et peut offrir des services.

Le comportement d'un agent résulte de ses perceptions, de ses connaissances, de ses compétences, et bien évidemment de son but. Il suit un *cycle de vie* en trois étapes se répétant indéfiniment tout au long de son exécution :

- la phase de perception durant laquelle l'agent acquiert de nouvelles informations sur l'environnement
- la phase de décision durant laquelle l'agent choisit les prochaines actions à effectuer
- la phase d'action durant laquelle l'agent réalise les actions décidées à l'étape précédente.

La propriété la plus importante d'un agent est son autonomie. Celle-ci se rapporte au contrôle de son exécution : un agent peut dire "non" à une requête, il décide lui-même d'agir ou non, et de la nature de ses actions. C'est d'ailleurs ce qui le différencie d'un simple sous-programme. Cette autonomie sous-entend qu'un agent a des buts qui lui sont propres, et qu'il peut les privilégier à la résolution d'une requête extérieure.

Enfin, notons que cette définition s'appuie fortement sur la notion d'environnement, qui elle ne bénéficie pas d'un consensus dans la communauté. Intuitivement, l'environnement peut être décrit comme tout ce qui est extérieur à l'agent et qu'il peut percevoir. Nous reviendrons sur ce point un peu plus tard, dans la section 3.1.2.2.

3.1.1.1 Différents types d'agents

Outre l'autonomie qui est une propriété commune à tous, les agents possèdent d'autres caractéristiques par lesquelles ils peuvent être différenciés (Gleizes et al. 2011).

Réactif Un agent est dit *réactif* lorsqu'il est capable de réagir aux modifications de son environnement. Ses actions sont déclenchées par les évènements dans l'environnement. On parle alors de *comportement réflexe*. Les conditions des règles de comportement reposent sur ses perceptions et son état interne. Un agent réactif n'a généralement pas (ou très peu) de mémoire.

Proactif À l'opposé des agents réactifs se situent les agents *proactifs*. De tels agents sont capables de modifier leurs buts et d'en générer de nouveaux. Ils disposent d'une mémoire et mettent en œuvre des algorithmes d'apprentissage complexes. On s'y réfère parfois sous le nom d'agents *cognitifs*. C'est le cas par exemple des agents Croyance-Désir-Intention (BDI, pour *Belief-Desire-Intention*, Rao et Georgeff 1995). Il n'y a en réalité pas de limite franche entre agent réactif et agent proactif, ils représentent les extrémités d'une échelle de granularité. Les agents réactifs, moins complexes, sont généralement présents en plus grand nombre dans

un système et chacun d'entre eux s'occupe d'une tâche relativement simple. On parle d'un système à granularité fine. Les agents cognitifs sont quant à eux de granularité plus grosse, chacun peut prendre en charge une tâche plus compliquée et ils sont donc généralement moins nombreux pour former un système.

Situé Un agent est qualifié de *situé* lorsque ses perceptions et sa communication avec les autres agents sont conditionnées par son positionnement dans l'environnement. Par exemple, un agent-piéton dans une simulation de foule ne percevra pas les mêmes obstacles selon sa position. C'est un agent situé. De même, un agent-fourmi qui communique avec les autres au travers de l'environnement (en déposant des phéromones) est un agent situé.

Communicant Les interactions des agents *communicants* ne dépendent pas de leur positionnement. Leur environnement ne possède pas de référentiel, et les agents communiquent directement par envoi de messages sans qu'une quelconque mesure de position n'interfère. Là encore, la frontière entre situé et communicant n'est pas absolue. Des agents situés peuvent utiliser divers canaux de communication, dont l'envoi direct de messages.

D'autres caractéristiques sont parfois associées aux agents. Notons par exemple les agents *sociaux* qui tiennent compte des autres agents à divers degrés dans leur raisonnement, ou encore les agents *adaptatifs* qui modifient leur comportement au cours de leur vie.

3.1.1.2 Architecture d'un agent

Les agents possèdent généralement une architecture modulaire bien définie. Les modules utilisés varient selon les applications, mais trois se retrouvent systématiquement :

- Le module perception prends en compte les évènements provenant de l'environnement. Il peut s'agir d'une simple boîte à lettre dans le cas d'un agent communicant, ou d'un module gérant des capteurs pour un agent situé.
- Le module décision implémente le raisonnement de l'agent. Il comprend les connaissances de l'agent et des règles pour les exploiter, et décider d'action à entreprendre en fonction du but de l'agent. Il peut s'agir de simples règles conditionnelles pour un agent réactif comme d'algorithmes d'apprentissage et d'un moteur d'inférence pour un agent cognitif.
- Le module action réalise les actions que le module décision lui demande. Il peut s'agir de simples envois de messages dans le cas d'agents communicants comme d'actions plus complexes mettant en œuvre des effecteurs. C'est ce qui est vu depuis l'extérieur comme le comportement de l'agent.

La figure 3.1 schématise ces modules et leur interaction avec l'environnement. Cette architecture de base est souvent enrichie de modules propres à l'approche, par exemple d'un module spécifique pour les connaissances, les compétences ou encore l'apprentissage. De nombreuses implémentations sont en fait hybrides, possédant des caractéristiques à la fois réactives et cognitives et des modules d'interaction à la fois situés et communicants. Cette modularité fait

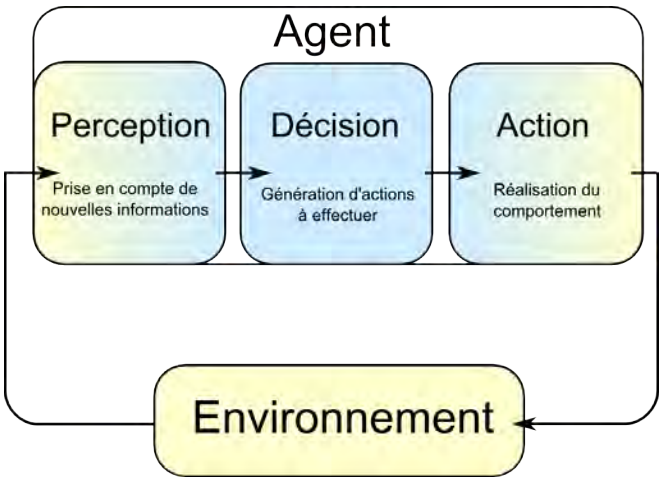

Figure 3.1 – Un agent et son environnement.

que la programmation par composants est particulièrement appropriée au développement d'agents, et donc de SMA (Noël 2012).

La notion d'agent étant maintenant bien définie, il est temps de s'intéresser aux systèmes multi-agents.

3.1.2 Qu'est-ce qu'un système multi-agent ?

Un système multi-agent (SMA) est un ensemble d'agents en interaction dans un environnement commun, agissant pour résoudre une tâche commune et cohérente. Ce dernier point est important car il implique l'unité du SMA, malgré le fait que chaque agent possède son propre but individuel qui peut éventuellement entrer en conflit avec celui des autres.

Cette composition particulière donne aux SMA des propriétés très intéressantes.

3.1.2.1 Propriétés des SMA

Tout d'abord, au sein d'un SMA, la connaissance et le savoir-faire sont distribués parmi les agents. Chaque agent pris individuellement possède sa propre représentation de l'environnement et ses propres compétences, qui sont insuffisantes pour accomplir la tâche globale du système. Mais toutes les connaissances et compétences nécessaires à la tâche sont malgré tout présentes, distribuées dans le système. Cette propriété importante rend les SMA particulièrement adaptés aux problèmes présentant une distribution naturelle.

Un SMA est autonome, aucun système ne le contrôle depuis l'extérieur. Cela provient directement de l'autonomie de chacun des agents. Ainsi, le contrôle d'un SMA est également décentralisé : chaque agent est responsable de son exécution. De ce fait, les agents d'un SMA s'exécutent généralement en parallèle et de manière asynchrone.

Un SMA est dit *ouvert* si des agents peuvent apparaître ou disparaître. Dans le cas contraire, le système est qualifié de *fermé*. Du fait de l'autonomie, la création d'un nouvel

agent est le plus souvent décidée par un agent existant du système. La disparition peut quant à elle être un suicide, ou bien être provoquée par l'environnement de l'agent selon son degré d'autonomie.

Enfin, un SMA dont tous les agents possèdent le même type de perception et d'action, et les mêmes compétences, est *homogène*. Si tous les agents ne partagent pas les mêmes capacités, le système est *hétérogène*. Par exemple, dans la modélisation multi-agent d'une équipe de basketball, chaque agent-joueur dispose du même type de perceptions (vision, ouïe) et d'actions (dribbler, passer, tirer, etc), seules certaines caractéristiques propres les diffèrent (taille, vitesse, etc). Le SMA est homogène. Mais si l'on ajoute le coach à ce modèle, avec des types d'actions différents (remplacements, temps-morts, consignes, etc), le système est hétérogène.

Les SMA se sont développés suivant trois branches principales : la résolution collective de problèmes, la simulation, et l'interaction avec l'utilisateur. La différence entre les trois tient en fait à leur environnement, plus ou moins contraint et faisant intervenir ou non un humain. Aussi, l'environnement est un élément essentiel de tout SMA qui se doit d'être maintenant abordé.

3.1.2.2 L'environnement

L'environnement est un concept très important lorsque l'on parle de systèmes multi-agents. C'est de lui que le système tire de l'information, et c'est lui qui supporte l'activité des agents. Il n'existe cependant pas de définition qui fasse consensus (Weyns et al. 2005).

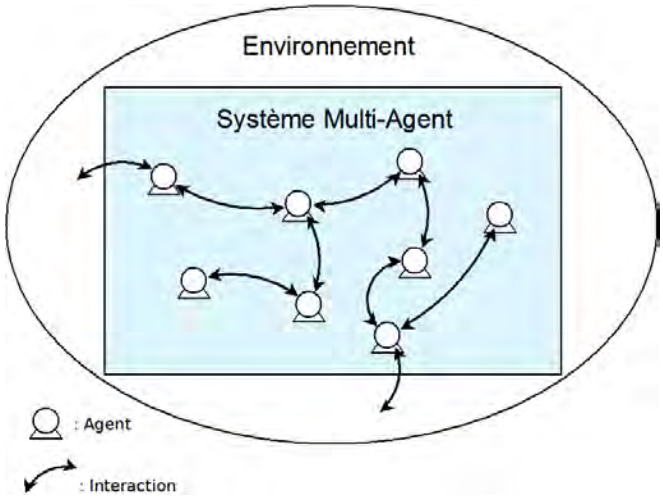

Figure 3.2 – Un système multi-agent et son environnement.

Différents environnements peuvent être identifiés selon le point de vue adopté. L'environnement du SMA comprend tout ce qui lui est extérieur. Par exemple, l'environnement d'une équipe de basket est constitué du terrain, du panier, de la balle, des adversaires, du public, etc, mais pas des joueurs qui la composent. De même, l'environnement d'un agent comprend

tout ce qui lui est extérieur, mais cela inclut les autres agents du système. L'environnement d'un joueur est celui de l'équipe, plus tous ses coéquipiers. Dans un SMA hétérogène, tous les agents n'ont pas les mêmes capacités de perception. Ainsi, il est possible que seule une partie des agents d'un SMA interagissent avec l'environnement du SMA, les autres étant internes au système (figure 3.2).

Les interactions entre un SMA et son environnement sont couplées. Lorsque le SMA effectue une action sur l'environnement, celui-ci est modifié. La perception de cette modification agit comme un signal de retour sur le SMA. Notons que l'agent qui effectue l'action et celui qui perçoit la modification ne sont pas forcément les mêmes.

On peut distinguer plusieurs facettes de l'environnement d'un agent. Il y a d'un côté l'environnement physique, composé de ce que les capteurs des agents peuvent percevoir, et de l'autre côté, l'environnement social, composé des autres agents connus. Dans l'exemple de la simulation d'une équipe de basketball, l'environnement physique est par exemple composé des paniers et de la balle, et l'environnement social des autres joueurs et du coach.

Enfin, il existe une variété d'environnements aussi vaste que les applications le permettent. L'environnement peut être continu ou discret, déterministe ou non, statique ou dynamique Russell et Norvig 2010). Autrement dit, il n'y a pas de restriction sur la nature de l'environnement pour y plonger un SMA, autre que celle de l'existence de moyens de perception et d'action.

La création d'un SMA demande donc de définir son environnement et de fournir un moyen d'interaction avec celui-ci. D'autres éléments sont nécessaires et sont décrits dans les paragraphes suivants.

3.1.2.3 Composition d'un SMA

Les agents et l'environnement sont les composantes les plus visibles d'un SMA, mais d'autres sont tout aussi cruciales. Demazeau 1995 propose par exemple une vue des SMA en quatre composantes élémentaires :

- Les agents, incluant la description complète de leur architecture et de leurs comportements.
- L'environnement, incluant la description des ressources avec lesquelles peuvent éventuellement interagir les agents.
- Les interactions, incluant l'ensemble des moyens assurant les interactions entre les agents, en particulier les protocoles de communication et les langages utilisés. Nous avons vu que les agents pouvaient communiquer soit par le biais de modifications sur l'environnement, soit directement par envoi de messages. La spécification de ces communications va avoir un impact important sur le comportement global du système. On regroupe sous le nom d'infrastructure les protocoles et les services permettant aux agents de fonctionner, assurant notamment leur communication. Ceux-ci ont fait l'objet de nombreux travaux, un des plus connus étant KQML, un langage permettant de manipuler des connaissances (Finin et al. 1994).

– Les organisations , incluant les éléments permettant de structurer un ensemble d'agents, par exemple sous forme de hiérarchie ou d'un simple réseau de relations. La section 3.2 revient plus longuement sur la notion d'organisation.

Cette approche se nomme *Voyelles*, du fait des initiales des quatre composantes.

Inclure son environnement dans la description d'un SMA peut paraître paradoxal, puisque celui-ci est censé être tout ce qui est à l'extérieur du SMA. En fait, cela montre le fort couplage qui existe entre les deux. À l'inverse des programmes classiques, un agent, comme un SMA, n'est pas conçu pour s'arrêter après un certain temps et donner un résultat. Il n'est utile que par son couplage avec l'environnement, sans lui, il ne peut ni percevoir, ni agir (Odell et al. 2003).

Au sein d'un SMA, les agents peuvent acquérir une caractéristique très intéressante : la localité. Un agent peut être conçu de sorte qu'il suive son propre but local, sans connaître la tâche globale du système dont il fait partie. Ses perceptions et ses actions sont également locales et l'activité globale du système résulte alors de l'ensemble des actions locales des agents. Cela permet de n'avoir à spécifier entièrement ni le problème (qui pourrait être trop complexe pour le permettre), ni sa solution (qui n'est de toute façon généralement pas connue). Ceux-ci sont répartis dans les buts et les comportements de chaque agent.

Certains SMA peuvent être analysés de manière récursive. Autrement dit, un agent peut être lui-même un SMA composé d'agents de niveau inférieur. Par exemple, Hoang, Occello et Jamont 2011 s'appuient sur l'approche Voyelles pour exprimer la récursion dans les SMA sous forme de règles de transformation de chaque composante pour passer d'un niveau à l'autre. Ils définissent également un modèle d'architecture d'agent, appelé SMA-R, permettant de mettre en œuvre différents niveaux de récursifs lors de la conception d'un SMA.

Enfin, la conception d'un SMA est généralement très proche du domaine d'application. Les agents correspondent à des éléments du problème, ce qui facilite l'instanciation du SMA. Il n'est en effet nul besoin de transformer les données du domaine dans une autre représentation (à l'image d'un génotype par exemple) pour qu'elles soient exploitables par le système.

3.1.3 Des applications de SMA

Les paragraphes suivant survolent quelques techniques de contrôle et d'apprentissage à base de SMA. Étant naturellement décentralisés, c'est dans les systèmes physiquement distribués qu'ils ont trouvé le plus d'applications. Mais leurs avantages ne se limitent pas à ce type de problème.

3.1.3.1 SMA et contrôle de systèmes

Le contrôle de chaînes de production fait intervenir des entités physiquement distribuées (les machines, les opérateurs, les produits, etc). Les systèmes de contrôle centralisés présentent le défaut majeur d'être rigides et de tomber dans des situations où toute la chaîne doit être stoppée lorsqu'une avarie y apparaît localement. C'est notamment pour cette raison que

les SMA et leur fonctionnement décentralisé sont régulièrement utilisés (Leitão 2009). Par exemple, Heragu et al. 2002 proposent unn SMA en trois couches :

- La couche "haute" n'est composée que d'un agent, appelé *Système* qui est responsable de la planification pour l'ensemble de l'usine. Il génère des buts pour chaque département.
- La couche "intermédiaire" est composée d'agents *Cellule*, chacun est responsable d'un département de l'usine. Ils transforment les objectifs fournis par la couche supérieure en buts pour les agents de la couche inférieure.
- La couche "basse" est composées d'agents de trois types, qui représentent les entités de l'usine impliquées dans la réalisation des objectifs : les *Consignes*, les *Machines* et les *MHD* (dispositifs de manutention).

Les agents de chaque couche négocient leurs activités en fonction de leurs ressources et selon un système d'enchères. Ils utilisent notamment une base de connaissances et un module d'apprentissage (incorporant un réseau de neurones). Ce processus de décision décentralisé permet une plus grande souplesse face aux imprévus, comme des pannes ou des commandes urgentes.

Les SMA peuvent être utilisés à divers niveaux dans le contrôle. Dans le domaine des bioprocédés, Gao et al. 2010 les utilisent à la fois pour concevoir le futur système contrôlé, et pour donner des conseils à un superviseur humain. Le SMA visant à proposer des actions de contrôle possède une structure hiérarchique dont le sommet est un agent coordinateur, suivi d'agents Opérations représentant les opérations spécifiques envisageables sur le bioprocédé, puis d'agents IA et Principes qui permettent l'utilisation de modèles prédictifs. L'avantage des SMA est ici encore la distribution. Elle permet le passage à l'échelle en mettant en relation plusieurs modèles couvrant chacun une échelle ou une plage de fonctionnement limitée du procédé. Aucun modèle global n'est en effet en mesure de représenter efficacement le procédé.

Dans ces approches, le contrôle est effectivement distribué. Le passage à l'échelle et la flexibilité sont assurés par la décentralisation des agents. Les agents spécifiques au domaine (Machines pour les chaînes de production, Opérations et IA pour les bioprocédés, etc) rendent l'instanciation plus facile, mais font perdre en généricité. Il faudrait les revoir entièrement pour appliquer le SMA à un autre type de système contrôlé. Des agents spécifiques au problème du contrôle, indépendamment du domaine, seraient préférables. En outre, l'apprentissage est réalisé par des techniques classiques, comme les réseaux de neurones, embarquées dans chaque agent. Ce point est susceptible d'alourdir leur déploiement sur des cas réels.

Certains SMA sont quant à eux entièrement dédiés à l'apprentissage.

3.1.3.2 SMA et apprentissage

Ce paragraphe se concentre sur des approches d'apprentissage multi-agent, à ne pas confondre avec des techniques parfois qualifiées de "basées agent" qui n'impliquent qu'un seul agent apprenant (par exemple sous la forme d'un LCS). Les techniques d'apprentissage multi-agent peuvent se séparer en deux catégories (Panait et Luke 2005) :

- L'apprentissage concurrent, où de multiples apprentissages sont menés simultanément, chaque agent embarquant le plus souvent entièrement un algorithme d'apprentissage.

Il y a donc plusieurs apprenants.
– L'apprentissage en équipe, où un seul apprenant découvre le comportement d'un collectif.

Apprentissage Concurrent L'apprentissage concurrent est l'approche la plus suivie. Chaque agent est un apprenant autonome, mais pas indépendant. Son apprentissage est influencé par son environnement, qui inclut les autres agents. En apprenant, un agent modifie son comportement (ses actions sur l'environnement) et peut ainsi potentiellement rendre invalide celui des autres agents. Ils vont apprendre, s'adapter et peut-être invalider à leur tour le comportement courant de l'agent. On parle alors de co-adaptation. Les algorithmes utilisés par les agents sont généralement directement dérivés du Q-learning et des algorithmes génétiques, et beaucoup font appel à des concepts de la théorie des jeux comme l'équilibre de Nash, par exemple Tuyls, Hoen et Vanschoenwinkel 2006.

En apparence, l'apprentissage concurrent peut se rapprocher du méta-apprentissage, avec l'idée de combiner efficacement plusieurs apprenants. Mais l'apprentissage concurrent n'est pas supervisé, et fait apparaître en réalité de nombreux challenges dus à la dynamique imprévisible de l'environnement. En particulier, la répartition du signal de renforcement entre les agents est un problème délicat et crucial pour le bon fonctionnement du SMA. Outre les limites des algorithmes utilisés par chaque agent, cela pèse sur le travail d'instanciation à fournir pour ce type de méthode.

Apprentissage en équipe L'apprentissage en équipe ne fait intervenir qu'un seul apprenant, mais celui-ci apprend le comportement d'un ensemble de plusieurs agents. Les méthodes utilisées diffèrent selon que le SMA soit homogène ou hétérogène. Dans le premier cas, un comportement est appris et appliqué à tous les agents. Dans le second, chaque agent est différent, permettant une plus grande richesse de comportements collectifs mais impliquant une plus grande complexité. Certaines approches hybrides combinent les deux en formant des groupes d'agents homogènes apprenant chacun des comportements collectifs différents (Bongard 2000).

Si le résultat est un SMA, l'apprentissage en équipe n'est pas lui-même multi-agent. Il s'agit en réalité d'une application presque directe de techniques d'apprentissage classiques. Les algorithmes génétiques sont par exemple utilisés en encodant le comportement de chaque agent dans le même long chromosome (Andre et Teller 1999). Cependant, cette approche met le doigt sur un aspect capital des SMA : le lien entre le comportement local des agents et le comportement global du système. En effet, l'apprentissage en équipe cherche à apprendre le comportement de chaque agent en se basant sur une évaluation du comportement global du système. Or, ce comportement global est quasiment impossible à prévoir à partir des comportements locaux des agents. C'est dans ce type de situation que le mot *émergence* est parfois évoqué. La section 3.2 donne plus de détails à ce sujet.

3.1.4 Bilan des SMA

Cette section a présenté les systèmes multi-agents et montré comment certaines de leurs propriétés pouvaient être bénéfiques pour le contrôle.

En particulier, la distribution des tâches permet de venir à bout d'une complexité plus importante que celle appréhendée par des méthodes classiques. Certains procédés sont trop complexes pour être modélisés entièrement ou permettre un apprentissage dans un temps acceptable. Les SMA permettent d'utiliser une collection de modèles plus simples, ou bien de poursuivre plusieurs apprentissages simultanément. Ils sont donc potentiellement capables de passer à l'échelle.

Les SMA sont capables d'apprendre, et leur fonctionnement par couplage avec un environnement les rend propices à l'apprentissage par renforcement. Ils paraissent donc intéressants pour le contrôle de systèmes dynamiques dont il faut suivre l'évolution.

Enfin, deux caractéristiques facilitent l'application des SMA. D'abord les agents utilisés sont généralement proches du domaine, ce qui facilite leur compréhension et leur intégration. Ensuite, l'autonomie des agents, et donc la décentralisation de la décision, assure une certaine résistance aux pannes (un agent défaillant ne met pas en péril tout le système). Cette modularité inhérente aux SMA permet également une grande flexibilité. Il est, par exemple, aisé de remplacer un agent par un autre (pour changer de modèle utilisé, pour changer une méthode d'apprentissage par une autre, etc) et de modifier les variables perçues ou contrôlées sans remettre en cause tout le fonctionnement du système.

Ces points sont cependant contrebalancés par le contenu des agents. L'utilisation de techniques complexes d'apprentissage, ou la présence de modèles et d'autres expertises liées au domaine compliquent la mise en place d'un SMA. Une solution pourrait provenir d'un apprentissage réellement multi-agent, c'est-à-dire porté par le comportement de tous les agents, et non par l'encapsulation d'une technique existante par l'un ou plusieurs d'entre eux.

Un tel apprentissage est possible si on donne la capacité aux agents de trouver eux-mêmes, localement, la meilleure organisation à adopter. Les sections suivantes approfondissent cette idée.

3.2 L'auto-organisation dans les SMA

Les SMA reposent sur l'idée, déjà évoquée dans ce document avec les réseaux de neurones, qu'un ensemble de petites entités relativement simples peut accomplir des tâches complexes. L'autonomie des agents, la richesse de leurs interactions, de leur comportement et de l'environnement dans lequel ils sont plongés poussent le concept encore plus loin. Cette section aborde le sujet de la relation entre l'activité locale des agents et la fonction globale du système.

3.2.1 La notion d'organisation

La fonction globale d'un SMA est réalisée par l'*organisation* des agents qui le composent. Avant d'aller plus loin, il est important de lever une ambigüité inhérente au terme "organisation". Celui-ci peut désigner aussi bien un processus (l'action d'organiser) qu'un état (le résultat de l'action d'organiser). Dans ce document, c'est le deuxième sens qui est utilisé. L'organisation décrit *qui* fait *quoi* et *comment* au sein du système, et non comment la réponse à ces questions à été décidée.

Dans le domaine des SMA, il existe une grande variété d'organisations et de façons de les implémenter. Ces différentes organisations contraignent de manière plus ou moins importante le comportement des agents, et donc brident quelque peu leur autonomie. Elles apportent en revanche une certaine assurance concernant la cohérence du comportement global du système développé. Les paragraphes suivants présentent quelques exemples de modèles d'organisation.

3.2.1.1 Modèles organisationnels

Modèle AGR Un des premiers modèles d'organisation d'agents a avoir été proposé se nomme AGR, pour Agent-Groupe-Rôle (Ferber, Gutknecht et Michel 2004). Il fait appel à trois composants dont il tire son nom :

- Agent : un agent peut assumer un ou plusieurs rôle au sein d'un groupe, et appartenir à plusieurs groupes différents. Aucune supposition n'est faite sur l'architecture interne de l'agent ni sur son niveau de complexité. Tout type d'agent peut donc être utilisé.
- Groupe : Deux agents ne peuvent communiquer entre eux uniquement s'ils appartiennent à un groupe commun. Les groupes réunissent des agents selon les tâches à accomplir.
- Rôle : Un rôle désigne la représentation de la fonction d'un agent au sein d'un groupe. Un agent peut jouer un ou plusieurs rôles dans le groupe, et un rôle peut être joué par plusieurs agents. Les rôles sont locaux dans chaque groupe.

Cet exemple de modèle d'organisation est un des plus répandus de par sa généricité et sa facilité d'application, ne faisant aucun présupposé sur la nature des agents et leur manière de communiquer avec les autres. Le concepteur d'un SMA utilisant AGR modélise les tâches à effectuer sous forme de groupes et de rôles, et c'est ensuite aux agents de s'engager dans un rôle et de le jouer. Il existe des extensions de ce modèle, notamment AGREEN qui intègre l'environnement dans la modélisation, et instaure des normes que les agents doivent respecter sous peine de sanctions (Báez, Stratulat et Ferber 2005). Bien que les agents puissent demander eux-mêmes la création d'un groupe, c'est une approche essentiellement *top-down* de conception de SMA.

Modèle OperA OperA est un modèle d'organisation un peu plus souple que AGR. Il spécifie des structures, des exigences et des objectifs tout en laissant aux agents la liberté d'agir selon leurs propres capacités et buts (Aldewereld et Dignum 2011). Le but est

d'apporter suffisamment de garanties sur le fonctionnement global du système tout en laissant de la marge afin d'intégrer des agents hétérogènes et de laisser place à une certaine adaptation. OperA est en fait constitué de trois modèles interdépendants :

- Le Modèle Organisationnel (MO) résulte de l'analyse du domaine d'application et décrit le comportement désiré pour l'organisation. Cela comprend la définition d'objectifs, de normes, de rôles, d'interactions et d'ontologies. Le MO doit disposer, à chaque instant, des agents adéquats pour chaque rôle. Il ne définit cependant pas de structure de groupe et ne contraint pas le comportement des agents.
- Le Modèle Social (MS) définit les règles de l'adoption d'un rôle du MO par un agent. Il décrit les accords que passe un agent lorsqu'il s'engage à remplir un rôle. C'est un système de contrat qui contraint le comportement de l'agent pendant la période durant laquelle il occupe un rôle.
- Le Modèle d'Interactions (MI) spécifie les accords entre les agents engagés dans des rôles afin d'assurer leurs interactions futures.

Ce modèle a besoin d'agents capables de décider quand rejoindre ou quitter une organisation et de passer des contrats. Il s'agit donc d'agents plutôt proactifs, et bien souvent d'agents BDI. Cela permet de laisser une grande part d'autonomie aux agents. La conception de SMA selon Opera est ainsi un mélange de *top-down* et de *bottom-up*. L'ouverture de l'organisation (c'est-à-dire le fait de pouvoir changer, durant l'exécution, les agents qui la composent) apporte à cette approche de bonnes capacités d'adaptation. Toutefois, concevoir l'organisation, et surtout les agents aptes à y prendre part nécessite une bonne expertise du domaine.

3.2.1.2 Bilan

Définir une organisation pour un SMA permet d'assurer son activité en terme de comportement global attendu. Cette assurance est obtenue au prix d'une autonomie réduite pour les agents qui doivent se conformer au rôle qui leur est attribué.

Selon la complexité de la tâche à effectuer par le SMA, il peut être très difficile, voire impossible, de définir une organisation (les rôles, mais aussi tous les différents contrats impliqués) capable de la réaliser. Une solution se trouve au croisement de l'apprentissage et de l'autonomie, en laissant aux agents le soin d'apprendre eux-mêmes la meilleure organisation.

3.2.2 Comportement local et fonction globale : auto-organisation et émergence

L'activité d'un agent peut être vue comme l'application d'une fonction. Dans ce cas, l'organisation d'un SMA est la composition des fonctions de ses agents. On a vu que l'organisation réalise la fonction globale du SMA. Apprendre cette fonction globale revient donc à trouver la bonne organisation. Cela peut se faire en modifiant le comportement d'un ou plusieurs agents, en modifiant les relations entre les agents, ou en ajoutant/supprimant des agents dans le système. Si ces ajustement relèvent de la responsabilité des agents eux-mêmes, il s'agit d'auto-organisation.

L'auto-organisation est une notion qui n'est pas exclusive aux systèmes multi-agents, mais qui s'applique dans bon nombre de systèmes complexes où une différence est faite entre niveau local et niveau global. De ce fait, c'est une notion qui est souvent rapprochée de celle d'émergence. Les paragraphes suivants présentent brièvement ces deux notions.

3.2.2.1 L'émergence

L'émergence est une notion centrale de l'étude des systèmes complexes, qu'ils soient naturels ou artificiels, mais qui n'a pas de définition formelle adoptée par tous. On qualifie souvent d'émergent un phénomène perceptible au niveau global résultant des interactions locales des parties du système (De Wolf et Holvoet 2005). Par exemple, la formation de colonnes lorsque deux larges groupes de personnes se croisent en sens inverse peut être considéré comme un phénomène émergent.

Cependant, limiter l'émergence à la séparation entre des interactions de niveau local (micro) et des observations au niveau global (macro) n'est pas suffisant. En effet, toute activité globale d'un système résulte des interactions entre ses parties. On ne considère généralement pas pour autant la rotation des aiguilles d'une montre comme émergente. D'autres critères sont à prendre en compte. Deux aspects paraissent particulièrement importants : le caractère dynamique du phénomène et le maintien de sa cohérence au fil du temps (Goldstein 1999, Heylighen 2001). Ainsi, un phénomène émergent apparaît au cours de l'évolution du système, il n'est pas formé au préalable, et il se maintient durant une période suffisamment longue pour avoir une identité propre.

À cela s'ajoute un troisième aspect qui est pour beaucoup au cœur de l'émergence : la *nouveauté* du phénomène émergent vis-à-vis du niveau micro (De Wolf et Holvoet 2005). Un phénomène émergent au niveau macro n'est pas prévisible depuis le niveau micro. Les entités du niveau micro n'ont aucune représentation explicite du comportement global et celui-ci ne peut pas être déduit depuis elles. De cela découle un autre critère de l'émergence, la décentralisation du contrôle : le niveau macro n'est pas directement contrôlable et aucune entité centralisée ne le maîtrise, ce contrôle passe toujours par les différentes entités du niveau micro. C'est pour désigner cette *nouveauté* que l'expression populaire "le tout est supérieur à la somme des parties" est parfois utilisée (Odell 2002). La définition assez peu évidente d'une "somme" appliquée aux parties d'un système complexe, ainsi que l'omission des interactions de micro-niveau (qui peuvent constituer ce "plus" dans le comportement du "tout" sans pour autant parler d'émergence) doivent nous pousser à mettre de côté cette formulation. Il demeure cependant pertinent d'appuyer sur le fait que l'émergence est caractérisée par l'impossibilité de prévoir ce qu'il advient du macro-niveau en ne se basant que sur l'étude du mirco-niveau. Lorsque l'on ne sait pas dérouler la chaîne de causalité entre les deux niveaux, c'est que l'on observe un phénomène émergent.

Ces dernières considérations mettent en lumière un aspect très important (et controversé) de l'émergence : elle dépend des connaissances de celui qui l'observe. Un observateur qui acquiert suffisamment de connaissances sur le système observé, au point de pouvoir dérouler

avec exactitude la chaîne de causalité entre les niveaux micro et macro, ne verra plus de phénomène émergent se former puisque l'aspect *nouveauté* sera manquant.

L'émergence peut-être en fait vue comme la compensation de notre incapacité à suivre la multitude d'interactions, potentiellement complexes, qui a lieu au niveau micro. Nous isolons naturellement des phénomènes au niveau macro afin de nous donner la capacité d'étudier un système donné. C'est là toute son utilité.

Notre question de départ concernait l'apprentissage, par une population d'agents, de la meilleure organisation en regard d'une fonction particulière trop complexe pour être spécifiée. L'émergence nous apprend qu'il n'est pas nécessaire pour les agents d'avoir connaissance du comportement global du SMA pour que celui-ci accomplisse la fonction désirée. Elle ne répond cependant pas entièrement à la question, car elle ne dit rien sur la manière d'aboutir à la fonction voulue. Il faut pour cela s'intéresser à l'auto-organisation.

3.2.2.2 L'auto-organisation

L'auto-organisation se retrouve dans de nombreux systèmes, naturels comme artificiels, et dans de nombreuses disciplines scientifiques, de la géologie à la sociologie, en passant par la chimie et la biologie, et bien sûr l'informatique. Cette diversité rend difficile l'établissement d'une définition claire englobant tous les aspects que peut prendre l'auto-organisation dans ces différents domaines. Intuitivement, l'auto-organisation est le fait qu'une structure ou une organisation apparaisse dans un système sans qu'un contrôle ou des contraintes ne soient imposés depuis l'extérieur (Di Marzo Serugendo, Gleizes et Karageorgos 2011). En d'autres termes, l'organisation courante d'un système auto-organisateur résulte de contraintes et de mécanismes internes à ce système, basés sur des interactions locales entre ses composants (Camazine et al. 2003). La nature dynamique de ces interactions rend le résultat de l'auto-organisation, le comportement global du système, souvent imprévisible. C'est ainsi que l'émergence et l'auto-organisation sont liées.

Nous nous intéressons ici à l'auto-organisation dans les systèmes logiciels. Elle concerne les programmes capables de changer leur organisation, afin de s'adapter aux changements de leur environnement ou de leurs buts, sans contrôle explicite externe. Di Marzo Serugendo, Gleizes et Karageorgos 2011 en proposent la définition suivante :

Définition : L'auto-organisation est le processus permettant à un logiciel d'altérer dynamiquement son organisation interne (structure et fonctionnalité), pendant son exécution, et sans l'aide d'un mécanisme explicite de contrôle externe.

Une différence est faite entre l'auto-organisation faible, pour laquelle le processus d'auto-organisation est assuré de manière centralisée à l'intérieur du système, et l'auto-organisation forte où le contrôle de ce processus est décentralisé. Il y a donc deux processus dans un système auto-organisateur : celui qui réalise l'auto-organisation, et celui qui réalise la fonction pour laquelle le système est conçu. Ils sont toutefois si entremêlés qu'il est difficile de dire si une interaction locale appartient à l'un ou à l'autre.

Un des aspects fondamentaux de l'auto-organisation est l'adaptation qu'elle procure. Dans un SMA, un changement de l'organisation signifie un changement de la fonction globale accomplie par le système. L'auto-organisation y est donc une forme d'apprentissage. La section suivante montre deux des mécanismes d'auto-organisation les plus utilisés dans les SMA.

3.2.3 Résoudre des problèmes grâce à l'auto-organisation

Résoudre un problème à l'aide de l'auto-organisation d'un SMA implique de modéliser le problème dans l'environnement. Il s'agit de créer une boucle de rétroaction, similaire à celle du contrôle en boucle fermée, qui pousse le SMA à s'ajuster en fonction du problème, et à agir en retour sur ce dernier (Simonin et Gechter 2006). Le SMA finit par se stabiliser dans une certaine organisation, correspondant à la solution du problème.

Les auteurs proposent une méthode générique pour la conception de SMA réactifs dédiés à la résolution de problème. Elle s'articule en quatre étapes principales :

- Définir un modèle du problème à résoudre, autrement dit l'environnement.
- Définir les perceptions des agents. En particulier, les agents doivent percevoir les aspects de l'environnement qui sont considérés comme des contraintes du problème.
- Définir les mécanismes d'interaction des agents, c'est-à-dire : des réactions locales aux éléments perçus du problème, des mécanismes permettant de résoudre les situations où ces réactions sont insuffisantes, et enfin des actions de régulations dans les cas où le SMA présente des risques d'instabilité.
- Mesurer/observer le résultat en tant que structure émergente du SMA. Cette structure ne peut être observée qu'au niveau global du système.

Cette méthode est intéressante car elle illustre comment l'influence de l'environnement sur les agents peut être mise à profit. Elle laisse néanmoins ouverte la question de l'évaluation de la stabilité du SMA, qui est cruciale pour différencier une solution d'une organisation intermédiaire. Ceci est un problème récurrent dans les SMA, particulièrement difficile lorsque l'environnement est dynamique. Enfin, elle est indépendante du modèle d'interaction qu'utilisent les agents. Autrement dit, elle supporte divers mécanismes d'auto-organisation mais ne guide pas strictement le choix du concepteur à ce sujet.

3.2.4 Mécanismes d'auto-organisation

Étant composés d'entités autonomes interagissant localement, les systèmes multi-agents sont particulièrement appropriés à l'utilisation de mécanismes d'auto-organisation. Ceux-ci sont le plus souvent directement inspirés de la biologie ou de la sociologie.

3.2.4.1 Stigmergie

La stigmergie est un processus d'auto-organisation par modification de l'environnement que l'on trouve par exemple chez les insectes sociaux tels que les fourmis, les termites ou les abeilles (Abraham, Grosan et Ramos 2006). Ce mécanisme permet la coordination

décentralisée d'agents, grâce à des règles simples, et sans que ceux-ci ne possèdent de connaissances globales (Bourjot, Desor et Chevrier 2011). Sa forme la plus courante est le dépôt de phéromones, une substance volatile dont la concentration locale guide les agents. C'est le principe de base de l'optimisation par colonie de fourmis (*ant colony optimization*, ACO) dans laquelle une population d'un nombre donné d'agents-fourmis cherche le plus court chemin en se déplaçant sur les arcs d'un graphe, procédant à des allers-retours entre un point de départ et un point d'arrivé communs prédéfinis. Chaque agent-fourmi dépose sur son passage des phéromones qui s'évaporent avec le temps (Dorigo, Di Caro et Gambardella 1999). Aux intersections, la probabilité d'emprunter un arc est d'autant plus grande que la concentration en phéromones y est forte. Ainsi, plus un arc est emprunté, plus il a de chances de l'être encore. Du fait de l'évaporation des phéromones, les arcs des chemins les plus courts sont privilégiés et le système converge vers un optimum. L'usage de probabilités laisse place à l'exploration de divers chemins et à une mise à jour éventuelle si le graphe évolue ou si le point d'arrivée change.

La stigmergie est parfois désignée sous le nom d'essaim intelligent, par analogie avec les insectes et parce que les agents sont homogènes. Elle est particulièrement adaptée aux problèmes d'optimisation discrète. Elle a également été appliquée à des problèmes de contrôle comme la régulation de la charge dans un réseau *peer-to-peer* (Montresor, Meling et Babaoğlu 2003) ou l'optimisation de chaînes de production (Valckenaers et al. 2007). Outre l'expression du problème sous forme d'un environnement situé dans lequel les agents peuvent se déplacer, la taille de la population, le taux d'évaporation des phéromones et le calcul des probabilités sont les principaux paramètres à définir pour chaque application.

3.2.4.2 Holons

Le terme holon a été introduit par Koestler dans un essai philosophique (Koestler 1967) comme une tentative de compromis entre holisme et réductionnisme. Un holon est une structure pouvant être vue à la fois comme une partie de holon d'un plus haut niveau (un super-holon) et comme un tout composé d'autres holons. Par exemple, un organe peut être vu comme une partie du corps humain, mais aussi comme un tout composé de cellules. Cette idée fut plus tard appliquée aux systèmes multi-agents : les agents abandonnent une partie de leur autonomie pour "fusionner" leur activité et ne plus être vus depuis l'extérieur que comme un seul agent, un holon (Gerber, Siekmann et Vierke 1999). Une modélisation en niveaux successifs de holons est appelée une holarchie (figure 3.3). Le système de Heragu et al. 2002, présenté dans la section 3.1.3.1 en est un exemple appliqué au contrôle de chaînes de production.

Une holarchie n'est pas nécessairement auto-organisatrice, mais certains mécanismes peuvent lui donner cette propriété. Par exemple, ADACOR (pour ADAptive holonic COntrol aRchitecture) utilise un principe de diffusion de l'information similaire à celui des phéromones, et s'applique au contrôle de chaînes de production (Leitão et Restivo 2006). Une autre possibilité d'auto-organisation et de permettre la division et la formation de holons pendant l'exécution. Par exemple, Rodriguez et al. 2011 proposent un modèle dans lequel

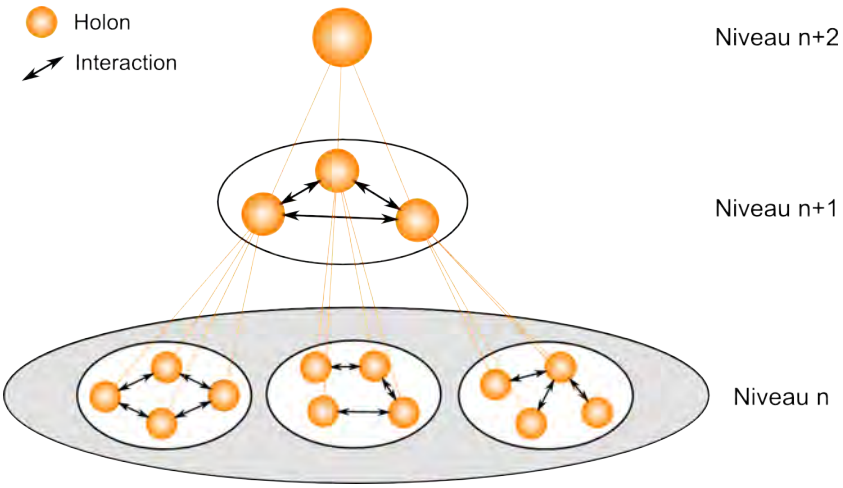

FIGURE 3.3 – Une holarchie à trois niveaux.

chaque holon d'un super-holon tient un rôle lié à l'auto-organisation. Chaque holon est soit une *partie*, soit la *tête* du super-holon. Il n'y a qu'une *tête* par super-holon, et celle-ci est responsable d'accepter ou de refuser de nouveaux holons dans le super-holon en cours de formation. Les règles d'acceptation doivent être définies en regard des objectifs du système, et certains holons peuvent éventuellement faire partie de plusieurs super-holons.

Dans de tels systèmes, chaque holon cherche à maximiser sa satisfaction, et doit pour cela se regrouper en super-holon avec d'autres agents. La satisfaction d'un holon comprend la satisfaction issue de ses propres efforts, celle découlant des autres holons avec qui il interagit, et celle relative au rôle qu'il tient. Les holons doivent également disposer d'une autre mesure, celle d'affinité, afin de choisir avec qui se regrouper. Cette mesure d'affinité indique le degré de compatibilité avec un autre holon, c'est-à-dire la possibilité ou non de coopérer avec lui pour accomplir un but commun. Elle est nécessairement liée au domaine.

L'auto-organisation permet ainsi à un système multi-agent holonique de s'adapter aux changements de son environnement, mais nécessite d'intégrer un comportement spécifique aux agents qui le composent afin qu'ils puissent coopérer.

3.2.4.3 Autres mécanismes

D'autres modèles d'auto-organisation dans les SMA existent, certains se basent sur la confiance et la réputation (DONDIO et al. 2006), d'autres sont inspirés des capacités d'adaptation du système immunitaire humain (ISHIDA 2004), et d'autres encore imitent la diffusion d'information (sous forme de rumeurs, ou de maladies) au sein d'une population (JELASITY et al. 2007). Tous mettent à profit la distribution et la décentralisation des SMA pour aborder des problèmes dynamiques auxquels l'auto-organisation permet de s'adapter.

3.2.5 Bilan de l'auto-organisation

Nous avons vu précédemment que la distribution et la décentralisation permettent à un SMA de s'attaquer à des problèmes complexes. Cette section a présenté deux notions importantes, l'émergence et l'auto-organisation, qui ont en commun de s'intéresser au lien entre le micro-niveau (les agents) et le macro-niveau (le système multi-agent). L'émergence nous apprend que les agents n'ont pas besoin d'avoir la connaissance de la tâche globale pour que le SMA l'effectue. L'auto-organisation permet à un SMA d'apprendre et de s'adapter à un environnement dynamique, et peut produire des comportement émergents. Et si cet environnement représente un problème à résoudre, l'auto-organisation permet de trouver des solutions (que l'ont peut éventuellement qualifier d'émergentes). Des méthodes mettant en œuvre des mécanismes d'auto-organisation ont été brièvement introduites. L'une d'elle, les SMA holoniques, soulève notamment un besoin crucial au bon déroulement d'un processus d'auto-organisation : les agents doivent coopérer.

Or, il se trouve que la coopération est justement la pierre angulaire de l'approche décrite dans la section suivante.

3.3 L'approche AMAS

Cette section présente la théorie des systèmes multi-agents adaptatifs (*adaptive multi-agent systems*, AMAS) développée par l'équipe SMAC. Les facultés d'adaptation d'un AMAS proviennent de sa capacité à s'auto-organiser, qui elle même repose sur l'attitude coopérative des agents qui composent le système (Georgé, Gleizes et Camps 2011). Ces propriétés, ainsi que les qualités intrinsèques des SMA, en font une approche particulièrement intéressante pour le contrôle de systèmes complexes. Les bases de cette approche sont exposées avant de s'intéresser à la conception et au développement de tels systèmes.

3.3.1 Interactions et coopération

Nous avons vu qu'il existe un fort couplage entre un SMA et son environnement, ils s'influencent réciproquement par une interaction permanente. L'activité d'un système sur son environnement peut être de trois types (Kalenka et Jennings 1999) :

- Coopérative : Les actions de l'un favorisent l'activité de l'autre. Les échanges entre système et environnement apportent des bénéfices mutuels.
- Neutre : Les actions de l'un n'entravent ni ne favorisent l'activité de l'autre.
- Antinomique : Les actions de l'un empêchent l'autre d'accomplir son activité.

Un système est en état coopératif lorsque ses actions sur l'environnement sont entièrement coopératives. Il est dans un état non-coopératif si certaines de ses actions sont neutres ou antinomiques. Ces définitions sont importantes pour caractériser l'activité d'un système, mais ne nous disent rien sur la fonction qu'il réalise, ni sur sa composition interne.

3.3.2 Adéquation fonctionnelle

Intuitivement, un système est fonctionnellement adéquat lorsqu'il exécute la fonction pour laquelle il a été conçu. C'est habituellement à un observateur extérieur que revient la possibilité d'évaluer l'adéquation fonctionnelle d'un système. Mais pour un SMA auto-organisateur, cette évaluation doit venir de lui-même, c'est-à-dire des agents qui le forment. Or ces agents ne connaissent pas la fonction globale. Ils doivent donc se référer à des critères purement locaux. Pour aider à trouver de tels critères, la théorie des AMAS stipule qu'un système dont les agents sont tous dans un état coopératif est fonctionnellement adéquat (Glize 2001). La démonstration repose sur une définition précise de l'adéquation fonctionnelle.

Définition Un système fonctionnellement adéquat n'a aucune activité antinomique sur son environnement.

Il découle de cette définition que tout système en état coopératif est fonctionnellement adéquat. Comme l'illustre la figure 3.2, l'ensemble des interactions entre un SMA et son environnement est en fait un sous-ensemble de l'ensemble des interactions de tous les agents. On en déduit que tout système à milieu intérieur coopératif (c'est-à-dire dont toutes les parties sont en état coopératif) est un système en état coopératif. Ainsi, tout système à milieu intérieur coopératif est fonctionnellement adéquat (figure 3.4).

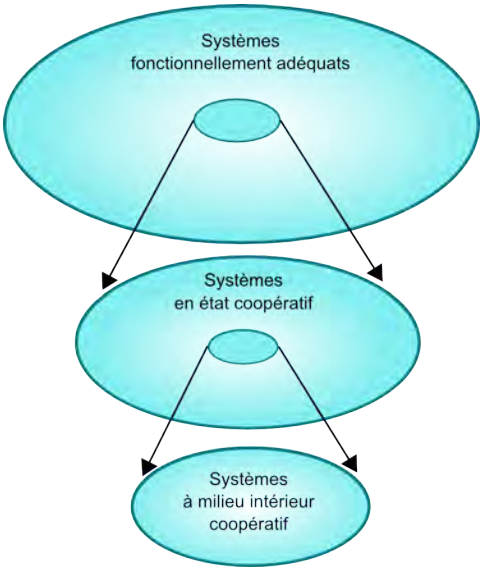

Figure 3.4 – Inclusion des ensembles de systèmes.

Il est également démontré que si un système fonctionnellement adéquat existe, un système équivalent à milieu intérieur coopératif existe également. C'est le théorème de l'adéquation fonctionnelle.

Théorème de l'Adéquation Fonctionnelle *Pour tout système fonctionnellement adéquat, il existe au moins un système à milieu intérieur coopératif qui réalise une fonction équivalente dans le même environnement.*

Ainsi, pour tout problème dont la solution est effectivement calculable (au sens de Church) il existe un SMA, dont tous les agents sont en état coopératif, permettant de le résoudre.

3.3.3 La coopération comme moteur de l'auto-organisation

La section précédente a succinctement expliqué pourquoi un système dont les agents sont tous dans un état coopératif est fonctionnellement adéquat. La difficulté pour un SMA est d'atteindre un état coopératif (et donc l'adéquation fonctionnelle), et de s'y maintenir alors que l'environnement est complexe et dynamique. Heureusement, et pour reprendre les mots de la socio-linguiste Deborah Tannen, la coopération n'est pas l'absence de conflit, mais un moyen de les gérer (Tannen 1999). Il faut donc doter les agents de mécanismes capables de les faire tendre vers un état coopératif. Autrement dit, il faut que l'auto-organisation des agents soit guidée par des mécanismes de coopération.

Dans un premier temps, chaque agent doit être capable (seul ou à l'aide d'autres agents) de savoir s'il se trouve en état coopératif ou non. En d'autres mots, il doit être capable de détecter toute situation impliquant une interaction neutre ou antinomique. Une telle situation est appelée une situation de non-coopération (SNC). Dans un second temps, chaque agent (seul ou à l'aide d'autres agents) doit être capable de résoudre ces situations de non-coopération.

3.3.4 Situations de non-coopération

Un agent est en situation de non-coopération lorsqu'il y a un défaut de perception, de décision, ou d'action. Pour aider leur identification, plusieurs types de SNC ont été définis (Georgé, Gleizes et Camps 2011) :

- Incompréhension : l'agent ne peut pas extraire d'information du signal perçu.
- Ambigüité : l'agent peut interpréter le signal perçu de plusieurs manières.
- Incompétence : l'agent ne peut parvenir à aucune décision à partir de ses connaissances actuelles.
- Improductivité : les décisions de l'agent n'aboutissent à aucune action.
- Concurrence : l'agent pense que son action aura les mêmes conséquences que l'action d'un autre agent.
- Conflit : l'agent pense que son action sera incompatible avec celle d'un autre agent.
- Inutilité : l'agent pense que son action n'aura aucune conséquence sur son environnement.

La résolution des SNC est réalisée localement par les agents, en modifiant l'organisation du système (autrement dit, en modifiant *qui* fait *quoi* et *comment*), c'est-à-dire en s'auto-organisant. Un agent dispose de trois moyens différents pour modifier cette organisation :

- Ajustement : modifier son comportement en ajustant ses paramètres internes.

- Réorganisation : changer ses relations avec les autres agents, c'est-à-dire arrêter d'interagir avec un agent donné, prendre contact avec un nouvel agent, ou bien modifier l'importance des relations existantes.
- Ouverture : Décider de se supprimer, ou bien de créer un nouvel agent.

C'est au concepteur que revient la tâche de définir les règles de résolution des SNC. Elles se présentent sous la forme de simple règles comportementales condition-action (la condition étant la détection d'une SNC donnée) dictant la conduite à tenir. Lorsqu'une SNC est détectée, les actions décidées par ces règles dites *coopératives* supplantent celles décidées par les règles habituelles, dites *nominales*. Le comportement d'un agent peut ainsi se distinguer en deux parties :

- le comportement nominal qui assure l'adéquation fonctionnelle lorsque l'agent est en état coopératif,
- le comportement coopératif qui assure de ramener l'agent dans un état coopératif lorsque celui-ci est en SNC.

La coopération s'opère dans un AMAS principalement par la résolution de SNC, mais également par une règle de conception simple : un agent doit toujours aider celui qui en a le plus besoin (il peut s'agir de lui-même). Par exemple, si plusieurs agents désirent une ressource unique, ce n'est pas le premier arrivé qui est servi, mais l'agent le plus critique.

Une méthode a été développée afin de guider le concepteur d'un AMAS dans ses choix, elle est présentée dans la section suivante.

3.3.5 Développer un AMAS

Par son approche locale de la définition d'un comportement collectif auto-organisateur, la conception d'un AMAS se détache de celui des SMA habituels. Capitalisant sur plusieurs années d'expérience dans le développement de tels systèmes, des outils ont été proposés par l'équipe SMAC afin de guider la tâche du concepteur et du développeur. La méthode de conception ADELFE et l'outil de spécification d'architecture et de génération de code MAY en sont deux exemples qui ont largement contribué aux travaux de cette thèse. Ils sont rapidement présentés dans les paragraphes suivants.

3.3.5.1 Concevoir un AMAS

ADELFE (Atelier de DEveloppement de Logiciels à Fonctionnalité Emergente) est un processus de développement basé sur le Rational Unified Process (RUP, Kruchten 2004) auquel sont ajoutées des activités spécifiques aux AMAS (Bonjean et al. 2013). ADELFE est composée de 21 activités réparties en 5 phases (figure 3.5) :

- L'étude des besoins préliminaires, dans laquelle un cahier des charges précis est établi avec le client. Ce cahier des charges spécifie ce que doit faire le système, il en définit les limites et les contraintes.
- L'étude des besoins finals se base sur la phase précédente pour définir les cas d'utilisation, les besoins (fonctionnels ou non) et leurs priorités sont organisés.

– L'analyse identifie et définit les agents du système.
– La conception détaille l'architecture du système en termes de modules, de sous-systèmes, d'objet et d'agents.
– La dernière phase consiste en l'implémentation du système.

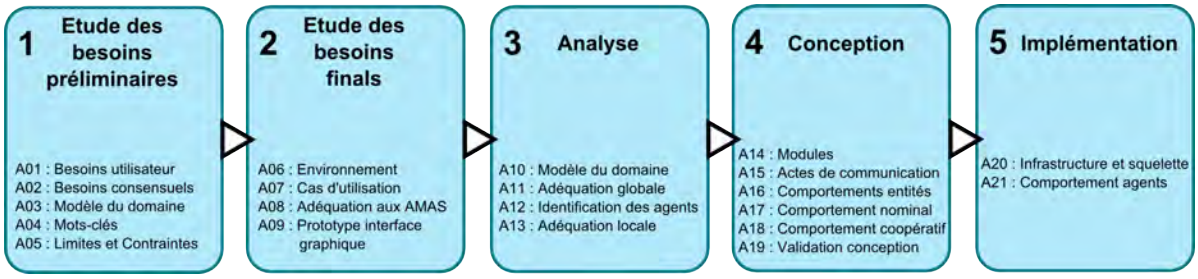

Figure 3.5 – Les cinq phases de la méthode ADELFE.

Notons qu'ADELFE n'est pas une méthode linéaire mais contient des bouclages et des processus incrémentaux dont la figure 3.5 ne rend pas compte, par souci de lisibilité. En plus des activités, ADELFE comprend la définition des termes liés aux AMAS, tels que SNC, environnement, etc. Elle propose également des lignes de conduite afin de faciliter la réflexion, notamment lors des étapes cruciales d'identification des agents (A12) et des SNC (A18).

Les phases d'analyse et de conception sont les plus spécifiques à l'approche AMAS. Lors de l'analyse, les activités 11 et 13 permettent de vérifier que l'utilisation des AMAS est pertinente en regard des besoins précédemment définis. Ensuite, lors de la phase de conception, l'objectif global du système est complètement mis de côté pour se concentrer sur le comportement purement local des agents. Cette approche locale, entièrement *bottom-up*, permet de contourner la complexité inhérente à la tâche globale que doit accomplir le système. Le concepteur se focalise en effet uniquement sur la fonction locale, plus simple, de chaque agent et ne cherche pas à anticiper le fonctionnement global du système qui sera émergent.

3.3.5.2 Implémenter un AMAS

L'implémentation d'un SMA nécessite la mise en place d'une infrastructure à-même de le supporter en termes d'exécution (gestion des ressources de calcul, etc) comme de comportement (mécanismes de communication, etc). Parce qu'elle facilite la réutilisation de code et qu'elle permet une certaine flexibilité, la programmation par composants est particulièrement adaptée.

L'outil MAY (Make Agent Yourself) permet de définir, dans un langage spécifique, l'architecture de l'infrastructure et des agents en termes de composants (Noël, Arcangeli et Gleizes 2012). L'outil génère ensuite un squelette de code JAVA que l'utilisateur n'a plus qu'à remplir. Plusieurs implémentations de composants sont proposées à la réutilisation, permettant une mise en place rapide des mécanismes de support génériques et laissant l'utilisateur se concentrer sur l'implémentation du comportement des agents.

3.3.6 Bilan de l'approche AMAS

Cette section a présenté une approche de conception de systèmes multi-agents adaptatifs (AMAS) basée sur la coopération. Un AMAS est un SMA dont les agents sont capables de détecter et de résoudre localement des situations de non-coopération. La résolution de ces situations dirige l'auto-organisation du système et conduit à une adaptation de la fonction globale. Les AMAS sont coopératifs dans le sens où ils disposent des mécanismes leur permettant de toujours revenir à un état où toutes leurs interactions sont coopératives.

Notons qu'un état de non-coopération (c'est-à-dire dans lequel certains agents entretiennent des interactions neutres ou antinomiques) n'empêche pas nécessairement l'émergence d'une fonction globale. Il met cependant gravement en péril l'adéquation fonctionnelle du système. Dans la pratique, il est possible que toutes les SNC ne soient résolues simultanément et définitivement. La nature dynamique de l'environnement provoque sans cesse l'apparition de ces situations et pousse ainsi les agents à s'auto-organiser et à s'adapter en permanence. De cette auto-organisation coopérative émerge une fonction globale, adéquate vis-à-vis de l'environnement, et dont les agents n'ont pas connaissance.

La section suivante analyse la pertinence d'une telle approche pour le contrôle (et son apprentissage) de systèmes complexes.

3.4 AMAS, contrôle et apprentissage

L'auto-organisation coopérative procure une excellente capacité d'apprentissage et d'adaptation aux AMAS et leur permet de s'attaquer aux problèmes complexes. Ils sont notamment utiles dans les cas présentant au moins une des caractéristiques suivantes (Georgé, Gleizes et Camps 2011) :

- il y a un problème à résoudre (une fonction à réaliser, une structure à observer, etc),
- l'application est complexe, dans le sens des systèmes complexes,
- le contrôle (du SMA) et l'apprentissage peuvent (et souvent doivent) être distribués,
- le système doit s'adapter à des changements, qu'ils soient internes (ajout/suppression de parties du système) ou externes (modifications dans l'environnement),
- les objectifs sont flous (impliquant des satisfactions humaines),
- le système est sous-spécifié, l'adaptation est dans ce cas un moyen de concevoir le système.

La problématique du contrôle de systèmes complexes correspond aux quatre premiers critères. En effet :

- le contrôle est une fonction à réaliser,
- la loi de la variété requise (Ashby 1956), présentée dans le chapitre 1, implique que le contrôleur doit être de complexité au moins égale à celle du système contrôlé,
- des techniques comme la commande prédictive distribuée (Müller, Reble et Allgöwer 2011) et l'auto-calibration en ligne (Malikopoulos, Assanis et Papalambros 2009) présentées dans le chapitre 2 renforcent l'idée de la nécessité de la distribution.

– un contrôleur de système complexe doit s'adapter aux changements de celui-ci au cours du temps, ainsi qu'à d'éventuelles pannes.

Le problème du contrôle est ainsi adéquat à l'utilisation des AMAS. En outre, ceux-ci semblent capables de répondre aux trois besoins exposés en fin de chapitre 2, rappelés ci-après.

Instanciation *Un système de contrôle doit être facile à appliquer à une instance particulière de procédé.* La difficulté d'application d'un système de contrôle à une instance de procédé provient généralement :

– de la construction et de la calibration d'un modèle,
– du besoin de connaissance et d'expérience sur les algorithmes d'apprentissage utilisés afin de les paramétrer correctement.

L'auto-organisation permet à un AMAS d'apprendre la réalisation d'une fonction. Il est alors tout à fait possible d'envisager de se passer totalement de modèle et d'apprendre directement la fonction de contrôle à partir d'observations sur les entrées et les sorties du procédé visé. En outre, la méthode ADELFE identifie des agents proches du domaine considéré. Les concepts et les données manipulés sont ainsi familiers des experts du domaine d'application. Il n'y a pas de transformation des données nécessaire à l'application du système (contrairement aux algorithmes génétiques) ni de connaissances poussées sur le domaine à intégrer au système sous forme de paramètres (comme c'est le cas pour calculer certains signaux de renforcement ou pour instancier un réseau de neurones).

Adaptation *Un système de contrôle doit être capable de suivre l'évolution du procédé, autrement dit d'apprendre parallèlement au contrôle afin de se mettre à jour.* Si l'auto-organisation permet de converger vers une solution, elle permet également de l'adapter dynamiquement. Ainsi, un système de contrôle AMAS devrait être capable d'ajuster son contrôle à mesure que son environnement (incluant le procédé contrôlé) change. Pour les mêmes raisons, il devrait être robuste vis-à-vis de pannes (de capteur ou d'effecteur), et éventuellement du bruit sur les données. Le fonctionnement par couplage avec l'environnement, proche de l'apprentissage par renforcement, permet cette adaptation en ligne, simultanément au contrôle.

Passage à l'échelle *Un système de contrôle doit être capable de passer à l'échelle, c'est-à-dire de gérer simultanément un grand nombre de variables contrôlées et de critères d'optimisation.* D'un point de vue calculatoire, la distribution et la décentralisation propres aux SMA rendent possible l'optimisation et la résolution de contraintes impliquant une combinatoire bien plus élevée qu'avec une approche centralisée. Elles permettent donc l'application d'un SMA au contrôle de systèmes de grande dimension. En outre, en se focalisant sur le niveau local et en laissant aux agents le contrôle de leur processus d'organisation, qui mène à l'accomplissement de la fonction globale du système, l'approche AMAS permet de repousser la limite de la complexité des systèmes ainsi conçus. Cette approche permet donc de concevoir plus facilement un contrôleur dont la complexité intrinsèque est plus importante que celle atteignable avec les méthodes usuelles. Or, selon la loi de la variété requise, le contrôleur d'un système complexe doit nécessairement être complexe.

Enfin, Videau 2011 a montré la pertinence de l'approche pour le contrôle de systèmes avec la conception d'un AMAS dédié au contrôle de bioprocédés. Celui-ci a servi de base de départ au système présenté dans le chapitre suivant.

CHAPITRE 4

ESCHER, contrôler et apprendre

Ce chapitre décrit et commente le système ESCHER. Acronyme de *Emergent Self-adaptive Controller for Heat Engine calibRation*, c'est un système multi-agent adaptatif pour le contrôle de systèmes complexes. Conçu dans le cadre d'un projet autour des moteurs à combustion, il se veut néanmoins suffisamment générique pour être applicable sans modification à une large gamme de procédés. Suivant une approche boîte noire du contrôle, il "joue" avec les entrées du procédé contrôlé et en observe les effets sur les sorties. Parallèlement, il déduit et applique les actions permettant de respecter les critères qui ont été fixés par l'utilisateur.

4.1 Objectifs du système

L'objectif premier d'un système de contrôle est de placer et maintenir le système contrôlé dans un état désiré. Dans le cas d'ESCHER, le système contrôlé peut posséder plusieurs entrées et plusieurs sorties (MIMO), et l'état désiré est décrit comme une combinaison de plusieurs critères. Un critère peut concerner aussi bien une variable unique (une sortie ou une entrée) qu'une combinaison de plusieurs variables, et peut être de trois types :

- une contrainte : un seuil à respecter,
- une consigne : une valeur à atteindre,
- une optimisation : une valeur à minimiser ou maximiser.

Une exigence pour ESCHER est d'être facile à mettre en place sur un procédé donné. L'utilisateur, qui est ici l'ingénieur appliquant ESCHER à un procédé donné, ne doit pas avoir besoin de connaître le fonctionnement de ESCHER pour l'utiliser, ni pour le paramétrer. Un contrôleur rapidement instanciable signifie, en outre, que son utilisation n'est pas soumise à un paramétrage lourd, ni à l'utilisation d'un modèle mathématique prédéfini. Autrement dit, les renseignements sur le procédé à fournir au contrôleur doivent être minimes. ESCHER doit ainsi se passer d'un modèle préétabli du système contrôlé et nécessiter peu de paramétrage. Il doit donc être capable d'apprendre le contrôle du procédé.

En outre, cet apprentissage ne doit pas être figé, il doit se poursuivre indéfiniment pendant le contrôle afin que ESCHER s'adapte aux changements de comportement du procédé (pannes, usure, etc). Pour apprendre, il se base sur l'observation en temps réel des entrées et des sorties du système contrôlé, ainsi que sur la connaissance (nécessaire à tout système de contrôle) des consignes, contraintes et objectifs définis par l'utilisateur. Ainsi, ESCHER voit le système contrôlé comme une boîte noire, il n'a pas connaissance des mécanismes internes qui en régissent le comportement.

Les sections suivantes présentent les agents qui composent ESCHER.

4.2 Comportement nominal

Dans l'approche AMAS, le comportement nominal d'un agent est celui qui lui permet de réaliser sa fonction lorsque toutes les conditions nécessaires sont réunies, c'est-à-dire lorsque il ne se trouve pas en situation de non-coopération (cf 3.3). Dans le cas d'ESCHER, cela équivaut au fait que le système a acquis un niveau de connaissance suffisant pour contrôler convenablement le procédé.

4.2.1 Tâches élémentaires d'un système de contrôle

Cette section présente un découpage en tâches élémentaires de l'activité de contrôle. Elle introduit progressivement les agents en charge de ces tâches, et décrit leur fonction ainsi que leur comportement nominal.

4.2.1.1 Observer le système contrôlé

Une des premières nécessités pour le contrôle d'un système avec une approche boîte noire est d'être capable de l'observer. Un type particulier d'agents sont en charge de la perception du procédé : les *Agents Variables*. À chaque entrée et sortie observables correspond un Agent Variable. Lors d'un cycle de vie, il perçoit la valeur de sa variable et la transmet aux autres agents qui ont besoin de cette information. Ainsi, la fréquence des cycles de vie des Agents Variables définit la fréquence d'échantillonnage des données. Un Agent Variable peut éventuellement embarquer un algorithme de filtrage de bruit si ce problème n'est pas géré par un système externe.

4.2.1.2 Représenter les critères de l'utilisateur

Le contrôleur doit avoir une représentation des objectifs de l'utilisateur. C'est la tâche des *Agents Critères*. Ceux-ci peuvent être de trois types :

- Seuil : l'agent exprime la volonté de maintenir une variable au-dessous ou au-dessus d'un seuil défini par l'utilisateur.
- Consigne : l'agent exprime la volonté de maintenir une variable à une valeur précise définie par l'utilisateur.
- Optimisation : l'agent exprime la volonté de minimiser ou de maximiser une variable.

Chaque Agent Critère reçoit les mises à jour de valeurs des Agents Variables qui le concernent, calcule un niveau de *criticité* et le transmet aux agents qui en ont besoin. Ce niveau de criticité traduit la satisfaction du critère représenté : plus il est élevé, moins le critère est satisfait.

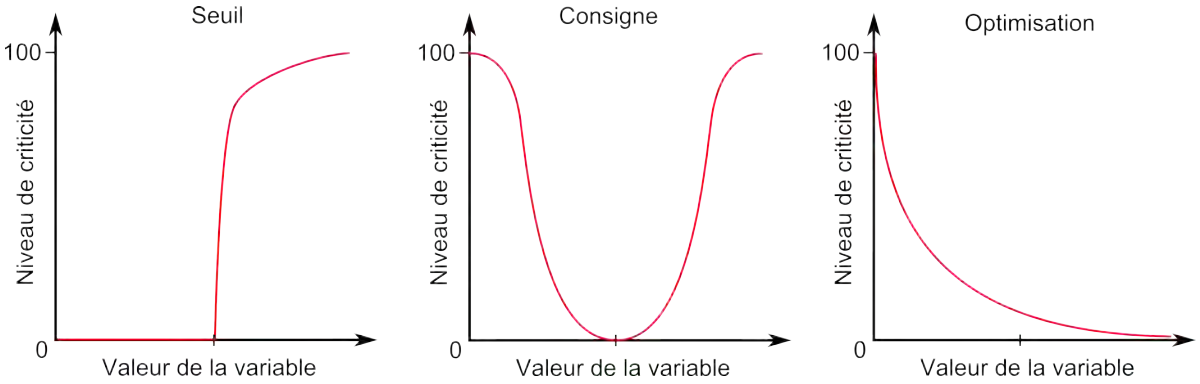

FIGURE 4.1 – Exemples de fonctions de criticité.

La figure 4.1 illustre des exemples de fonctions utilisées par chaque type d'Agents Critères pour le calcul du niveau de criticité. Lorsqu'il vaut 0, le critère est satisfait. À 100, la criticité est maximale, le critère est très loin d'être rempli. La fonction de criticité d'un seuil donne 0 lorsque le seuil et respecté, puis croît fortement jusqu'à atteindre le niveau maximal. Celle d'une consigne ne donne 0 que lorsque la consigne est atteinte et croît de part et d'autre de cette valeur (la symétrie n'est pas nécessaire). Enfin, un critère d'optimisation est traduit en niveau de criticité par une asymptote positive en 0 pour des variables non-bornées (ou par une fonction monotone non constante pour des variables bornées). Les formes de fonctions décrites ici sont modulables par l'utilisateur afin d'affiner l'importance relative de ses différents besoins, à la condition de ne pas présenter de minimum local et de se limiter à l'intervalle $[0;100]$ (voir 4.5).

Les Agents Critères opèrent une transformation de l'espace des variables du procédé vers l'espace des critères de l'utilisateur. Le niveau de criticité diminue à mesure qu'un critère sur les variables est en train d'être satisfait. Ainsi, les agents qui perçoivent les différents niveaux de criticité cherchent à les faire diminuer. Le seul moyen d'y parvenir est d'appliquer les actions adéquates sur les entrées du système contrôlé. Trouver les actions appropriées nécessite d'analyser l'état courant des variables et des critères, autrement dit de l'environnement du système, pour en trouver la dynamique.

4.2.1.3 Analyser l'état de l'environnement

L'environnement d'ESCHER est constitué du procédé contrôlé et des critères définis par l'utilisateur (figure 4.2). Grâce à l'ensemble des Agents Variables et des Agents Critères, ESCHER dispose d'une représentation interne, distribuée, de son environnement. Avant de pouvoir décider des actions à effectuer, il faut pouvoir analyser cet environnement en en tirer des informations pertinentes. C'est la tâche des *Agents Contextes*.

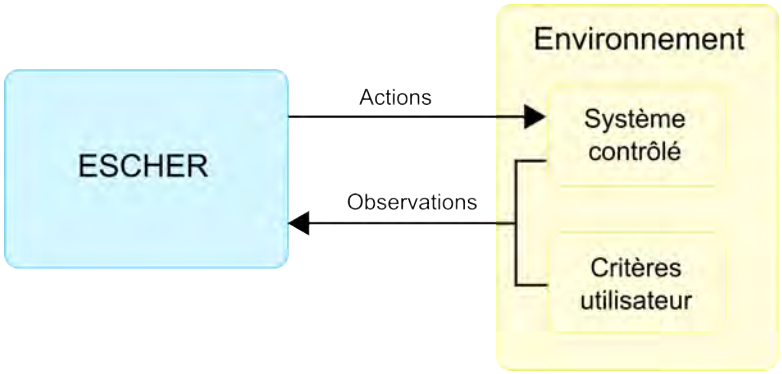

Figure 4.2 – ESCHER et son environnement.

Un Agent Contexte mémorise les effets sur chaque niveau de criticité d'une action appliquée sur un effecteur particulier. Il mémorise également l'état dans lequel se trouve l'environnement lorsque cette action est appliquée. Sa fonction est ainsi d'informer sur les conséquences attendues d'une action précise si elle est effectuée alors que le procédé se trouve dans un état donné.

Concrètement, un Agent Contexte est composé :

- d'une *action*, c'est-à-dire une modification sur une entrée du système contrôlé,
- d'un ensemble de *prévisions*, qui contient une valeur pour chaque Agent Critère représentant sa variation attendue de niveau de criticité,
- d'un ensemble de *plages de validité* représentant un état du procédé, contenant un intervalle de valeurs pour chaque Agent Variable.

Un Agent Contexte reçoit les mises à jour de valeur des Agents Variables ainsi que celles de criticité des Agents Critères. Lorsque la valeur courante de toutes les variables se trouve à l'intérieur de leur plage de validité, l'Agent Contexte est *valide*. Cela signifie que le système contrôlé se trouve dans un état dans lequel les prévisions de l'agent sont pertinentes. Lorsqu'un Agent Contexte devient valide, il envoie une notification contenant son action et ses prévisions, on appelle ce message une proposition d'action. Une notification est également envoyée lorsque l'Agent Contexte devient non-valide, et retire donc sa proposition. Ces messages sont reçus par l'*Agent Contrôleur* en charge de l'effecteur. Ce nouveau type d'agent est présenté dans la section suivante.

4.2.1.4 Appliquer l'action la plus adéquate

À chaque entrée contrôlée par ESCHER correspond un Agent Contrôleur. Sa fonction est d'appliquer l'action la plus adéquate sur cette entrée, c'est-à-dire l'action qui provoquera la plus forte diminution de niveau de criticité des Agents Critères. Une action peut être le fait d'incrémenter ou de décrémenter l'entrée d'un pas donné, ou bien de ne pas la modifier.

L'Agent Contrôleur base son choix sur les propositions qu'il reçoit de la part des Agents Contextes. Il maintient une liste des Agents Contextes valides, avec les actions proposées et les prévisions associées, et il choisit l'action correspondant aux "meilleures" prévisions. La

meilleure action à appliquer est celle dont les prévisions associées indiquent la plus forte diminution du niveau de criticité de l'Agent Critère le plus critique. Si aucune prévision n'indique de variation de cet agent, on s'intéresse alors au second plus critique, et ainsi de suite. Il effectue l'action sur l'entrée et informe de son choix les Agents Contextes valides ou précédemment sélectionnés. Il envoie ainsi :

- une notification d'acceptation aux Agents Contextes valides ayant proposé l'action sélectionnée,
- une notification de rejet aux Agents Contextes valides ayant proposé une action non sélectionnée,
- une notification d'abandon aux Agents Contextes dont l'action était jusqu'à maintenant appliquée.

Bien sûr, de nombreux cas surviennent dans lesquels un Agent Contrôleur n'est pas capable de prendre une bonne décision (c'est-à-dire une décision qui fera baisser les niveaux de criticité), à cause d'informations incomplètes ou incorrectes. Ce sont des situations de non-coopération (SNC). Elles apparaissent lorsque l'apprentissage d'ESCHER n'est pas complet et que celui-ci n'est pas encore pleinement adapté au procédé contrôlé. L'apparition d'une SNC déclenche un comportement spécifique des agents, le comportement coopératif, afin de la résoudre et de mettre ESCHER dans un état de fonctionnement adéquat. Les SNC et leur résolution sont présentées dans la section 4.3.

Pour le moment, nous poursuivons la présentation du système avec une vue générale de celui-ci et une illustration de son fonctionnement sur un cas simple.

4.2.2 Vue globale du système

ESCHER comporte donc quatre types d'agents, précédemment introduits :

- les Agents Variables sont les yeux du système, il y en a un par entrée et sortie du procédé ;
- les Agents Critères représentent les critères de l'utilisateur, l'état souhaité du procédé ;
- les Agents Contextes sont en quelque sorte la mémoire du système, ils représentent une portion de l'espace d'états de l'environnement dans laquelle les conséquences d'une action donnée sont connues ;
- les Agents Contrôleurs sont les mains du système, ils interagissent avec un ensemble d'Agents Contextes pour trouver la meilleure action à appliquer.

La figure 4.3 montre une vue globale du système, illustrant les liens entre les quatre types d'agents. Ceux-ci communiquent entre eux uniquement par envoi de messages.

4.2.2.1 Agents Contextes et Agents Contrôleurs

Chaque Agent Contrôleur est en relation avec un groupe d'Agents Contextes dont l'action mémorisée concerne l'entrée du procédé associée à ce même Agent Contrôleur. Il sélectionne la prochaine action à appliquer parmi les propositions qu'il reçoit et notifie de son choix les Agents Contextes qui s'étaient proposés. Il n'y a pas d'interaction directe entre les Agents

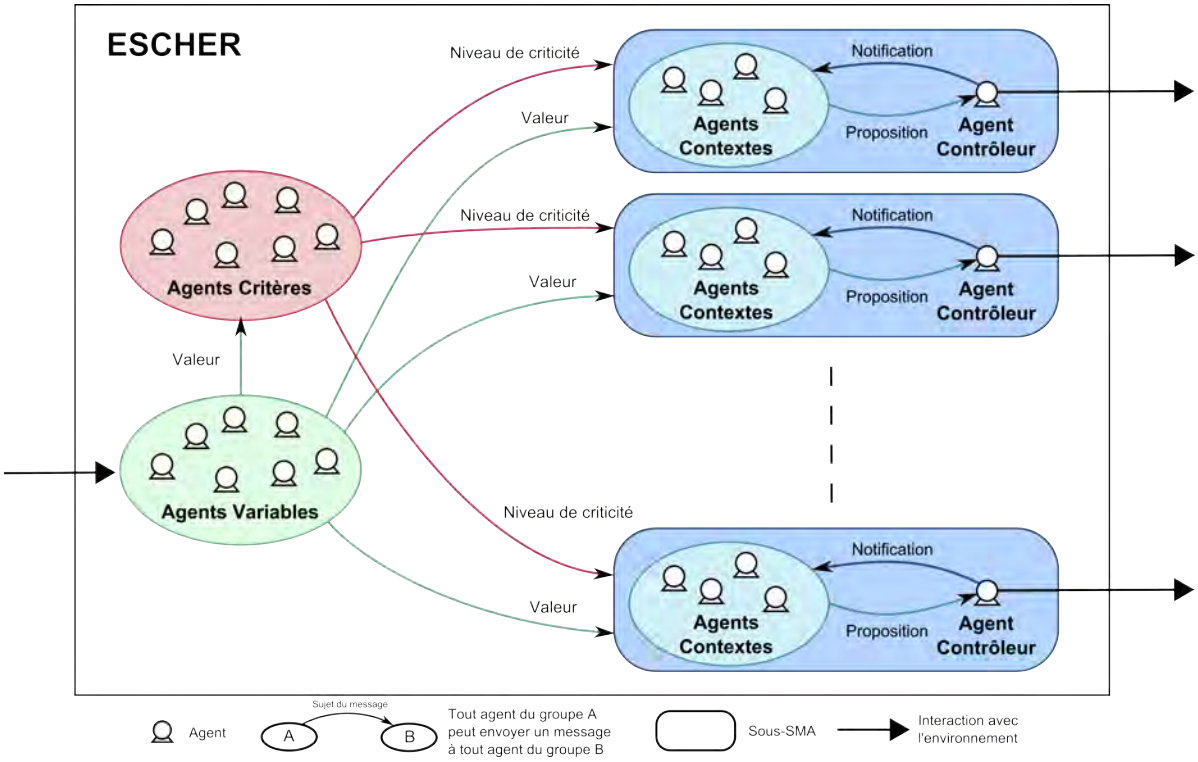

Figure 4.3 – Vue globale de ESCHER.

Contextes, ni entre les différents Agents Contrôleurs. Le seul lien entre ceux-ci passe par l'environnement : l'action d'un Agent Contrôleur va avoir des effets sur le procédé contrôlé, qui seront retranscrits par les Agents Variables et les Agents Critères et donc perceptibles par les autres Agents Contrôleurs.

Ainsi, un Agent Contrôleur et son groupe d'Agents Contextes peuvent être considérés comme un SMA à part entière, dont l'environnement est constitué des Agents Variables et des Agents Critères. C'est par l'observation des autres entrées du procédé qu'un "sous-SMA" Agent Contrôleur/groupe d'Agents Contextes synchronise les actions sur l'entrée dont il s'occupe avec celles appliquées sur les autres entrées (qu'elles soient le fait d'ESCHER ou non). Un Agent Contrôleur fait donc de son mieux pour faire diminuer les niveaux de criticité en s'occupant d'une seule entrée, localement, sans se soucier de comment sont contrôlées les autres. Il n'y a pas de processus de décision global pour trouver une action sur toutes les entrées à la fois. C'est cette caractéristique qui doit permettre à ESCHER de passer à l'échelle sur le nombre d'entrées contrôlées. En effet, si un Agent Contrôleur parvient à trouver les meilleures actions pour une entrée donnée d'un système MIMO, indépendamment des systèmes contrôlant les autres entrées, rien n'empêche que ces autres systèmes soient en réalité des Agents Contrôleurs. En outre, cette distribution du contrôle donne à ESCHER une certaine modularité, l'ajout d'un nouvel Agent Contrôleur n'impactant pas le fonctionnement des autres.

4.2.2.2 Agents Variables et Agents Critères

Pour accomplir sa fonction, chacun des agents, mis à part les Agents Variables, a besoin de connaître l'état courant du système contrôlé. C'est pourquoi ces derniers envoient leurs mises à jour de valeur à tous les autres types d'agents (les Agents Critères concernés, tous les Agents Contextes et tous les Agents Contrôleurs). Cette diffusion massive peut sembler problématique pour le passage à l'échelle, mais ce n'est pas le cas. En effet, les agents n'étant pas physiquement distribués, le coût d'un envoi de message est très faible. Au contraire, celui de la lecture de la valeur renvoyée par un capteur est important car il implique des systèmes externes, et donc probablement une communication réseau. Ainsi il est bien plus efficace d'avoir un agent par capteur, diffusant sa valeur aux autres, plutôt que chacun des agents récupérant les valeurs depuis l'extérieur du système.

Les Agents Critères transforment les valeurs des variables en niveaux de criticité reflétant la satisfaction des critères, autrement dit la correspondance entre l'état actuel et l'état souhaité du procédé. Ainsi Agents Variables et Agents Critères donnent à ESCHER une représentation complète de son environnement.

La section suivante déroule un exemple de contrôle dans un cas simple.

4.2.3 Illustration du fonctionnement

Cette section illustre un cas de contrôle d'une boîte noire simple (deux entrées et une sortie) par ESCHER. Nous nous intéressons ici au contrôle, et non à l'apprentissage, ainsi le système est supposé déjà adapté au système contrôlé.

4.2.3.1 Initialisation

ESCHER est initialisé avec autant d'Agents Variables qu'il y a d'entrées et de sorties sur le procédé contrôlé. La boîte noire considérée ici possède deux entrées (appelées E1 et E2) et une sortie (S1), chacune variant entre 0 et 1. Il y a donc dans cet exemple trois Agents Variables (respectivement VE1, VE2 et VS1). E1 et E2 sont arbitrairement initialisées à 0.05, plaçant la sortie S1 à 0.

Un Agent Contrôleur est créé pour chaque entrée contrôlée de la boîte noire. Dans notre cas, les deux entrées sont contrôlées par ESCHER, il y a donc deux Agents Contrôleurs, respectivement CE1 et CE2.

Les Agents Critères et leur fonction de criticité sont toujours à définir par l'utilisateur. Dans notre cas nous voulons placer et maintenir S1 à la valeur de 0,50. Un Agent Critère est donc créé avec une fonction de criticité de consigne (cf figure 4.1) nulle en 0,50, puis associé à l'Agent Variable VS1.

Enfin, pour ce cas particulier sans apprentissage, les Agents Contextes associés à chacun des Agents Contrôleurs sont faits à la main. La figure 4.4 représente ceux de l'Agent Contrôleur CE1. Nous y voyons l'action qu'ils proposent sur E1, leur prévision sur l'évolution du niveau de criticité de l'Agent Critère, et leurs plages de validité. Par exemple, l'Agent

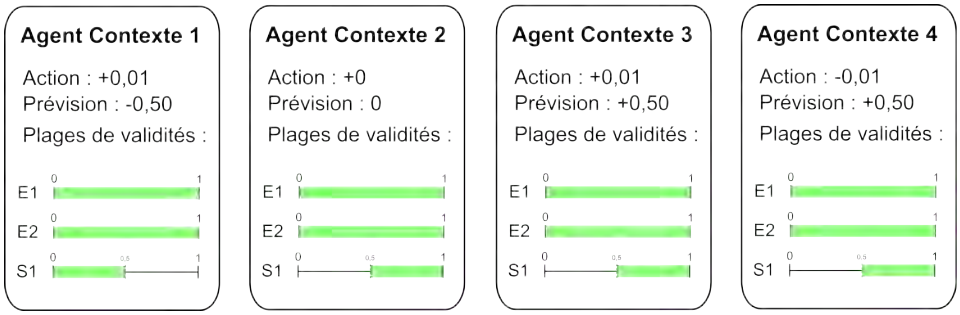

Figure 4.4 – Les Agents Contextes de l'Agent Contrôleur CE1.

Contexte 1 est valide lorsque S1 est inférieure à 0,5, peu importent les valeurs de E1 et E2 ; et non-valide sinon.

La figure 4.5 montre les agents de cette instance d'ESCHER et leurs liens de communication. Voyons maintenant comment se déroule leur activité.

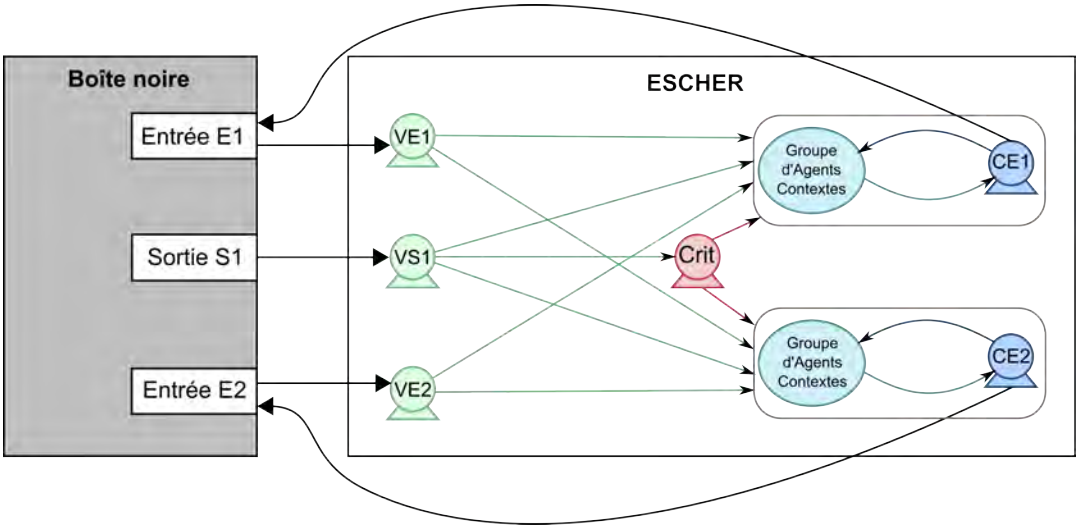

Figure 4.5 – Une instance de ESCHER pour une boîte noire simple

4.2.3.2 Communication entre les agents

Dans le cas général, chaque Agent Variable perçoit la valeur de sa variable correspondante de la boîte noire et la diffuse aux autres types d'agents. Les Agents Critères s'en servent pour calculer leur niveau de criticité, qu'ils diffusent aux Agents Contrôleurs et aux Agents Contextes. Chaque Agent Contexte vérifie si les valeurs des variables qu'il reçoit sont toutes à l'intérieur de leur plage de validité correspondante. Si oui, l'Agent Contexte est valide, et il envoie une proposition contenant son action et ses prévisions à son Agent Contrôleur associé. Chaque Agent Contrôleur trie les propositions reçues en fonction des prévisions qu'elles

contiennent, choisit celle qui prévoit la plus grande diminution de la criticité maximale, et informe les Agents Contextes concernés de l'acceptation ou du rejet de leur action.

Dans le cas particulier de notre exemple, à l'initialisation, les valeurs des variables sont $E1 = 0,05$, $E2 = 0,05$ et $S1 = 0$. Au niveau de l'Agent Contrôleur CE1, seul l'Agent Contexte 1 est valide et fait une proposition. Cette proposition indique une diminution du niveau de criticité. CE1 choisit et applique donc cette action. La criticité diminue effectivement et l'action est répétée tant que l'Agent Contexte 1 est valide et fournit la meilleure proposition. Dans notre cas cela correspond au moment où la sortie S1 atteint 0,5.

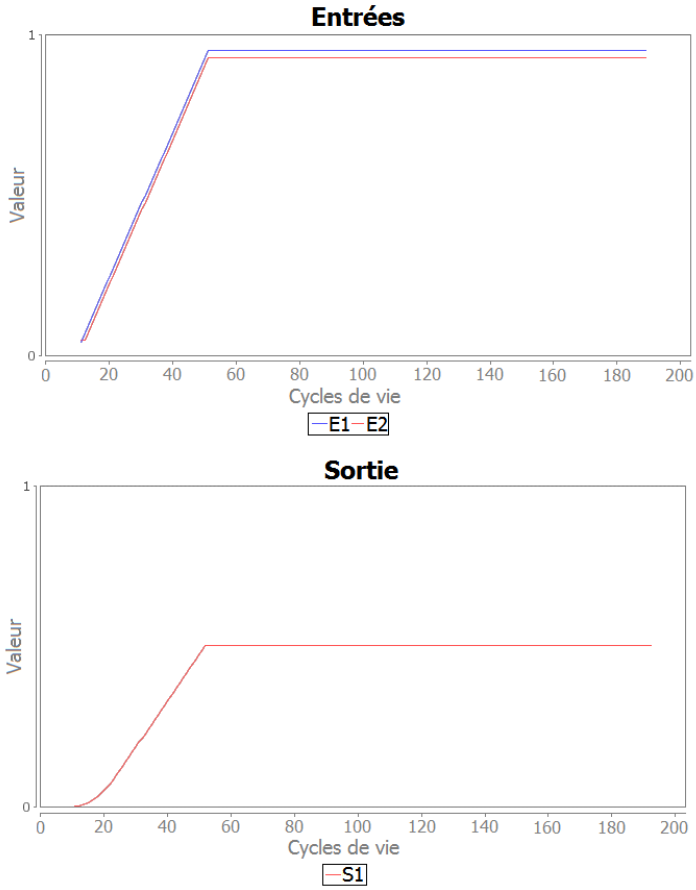

FIGURE 4.6 – Évolution des entrées et de la sortie de la boîte noire.

À ce moment, les Agents Contextes 2, 3 et 4 deviennent valides et proposent chacun une action. Deux d'entre eux (3 et 4) prévoient une augmentation de la criticité, c'est donc l'action de l'Agent Contexte 2 qui est sélectionnée. Il s'agit de ne pas modifier l'entrée E1, de manière à ne pas modifier la criticité. L'Agent Contrôleur envoie une notification de sélection de son action à l'Agent Contexte 2, de rejet aux Agents Contextes 3 et 4 et d'abandon à l'Agent Contexte 1. Comme les Agents Contextes ont été faits manuellement et correctement, cela arrive alors que la criticité est nulle. La sortie de la boîte noire a atteint la consigne et y est maintenue. L'Agent Contrôleur CE2 et ses Agents Contextes associés ont suivi une activité

similaire. La figure 4.6 montre l'évolution des entrées et de la sortie de la boîte noire de notre exemple. Les deux Agents Contrôleurs commencent par augmenter la valeur de leur entrée, provoquant la hausse de la sortie. Ils s'arrêtent lorsque cette dernière atteint la consigne fixée par l'utilisateur.

Les deux Agents Contrôleurs sont parvenus à faire baisser la criticité relative à une variable sur laquelle ils avaient tous les deux une influence, sans jamais se concerter explicitement, ni avoir recours à un modèle mathématique décrivant le comportement du système contrôlé. Cela a été possible grâce aux Agents Contextes, qui apportaient une certaine connaissance de la boîte noire.

4.2.4 Bilan du comportement nominal

Cette section a introduit les agents d'ESCHER et montré comment ils agissent pour contrôler ensemble un procédé. Les Agents Contrôleurs sont locaux, chacun s'occupe de son effecteur sans échanger directement avec les autres. Chacun est associé à un ensemble d'Agents Contextes, qui apportent des indications sur les actions à entreprendre.

La question est maintenant de savoir comment obtenir sans l'aide d'un humain des Agents Contextes adéquats, et ainsi pouvoir gérer des situations inconnues du système ? Bien sûr, cela se fait grâce à la résolution locale, par les agents, de situations de non-coopération. Celles-ci sont décrites dans la section suivante.

4.3 Situations de non-coopération

Cette section explique comment les agents résolvent les situations de non-coopération (SNC) qu'ils rencontrent. Puisqu'elles provoquent des changements dans l'organisation du système, les SNC et leur résolution sont la clé de l'adaptativité des AMAS. Chaque agent résout localement les SNC qu'il détecte par des actions spécifiques. Dans ESCHER, les SNC concernent principalement les Agents Contextes et les Agents Contrôleurs. Elles poussent le système à s'auto-organiser, notamment par la création, la modification ou la suppression d'Agents Contextes. Cette section s'intéresse aux conditions de déclenchement de cette auto-organisation. Les mécanismes mis en œuvre, par exemple pour l'ajustement de paramètres internes aux agents, relèvent de l'implémentation et sont présentés dans la section 4.5.

4.3.1 SNC 1 : Incompétence d'un Agent Contrôleur

Détection Quand un Agent Contrôleur n'a reçu aucune proposition d'action, il ne peut choisir une action adéquate avec certitude : il se trouve en SNC d'incompétence.

Résolution La résolution de cette SNC se déroule en deux étapes. Premièrement, l'Agent Contrôleur doit choisir lui-même une action. Il se base pour ça sur ses actions précédentes. Si la criticité est en train d'augmenter, l'action choisie est l'inverse de l'action précédente (par exemple une incrémentation au lieu d'une décrémentation). Sinon, l'action précédente

est répétée. Dans ce cas, si l'action précédente avait été proposée par un Agent Contexte maintenant non-valide, celui-ci ne recevra pas de notification d'abandon de son action. Enfin, si l'Agent Contrôleur n'a encore jamais appliqué la moindre action (c'est son premier cycle de vie), celle-ci est alors déterminée aléatoirement.

Si l'action ainsi choisie est la même que celle appliquée au cycle précédent, l'Agent Contrôleur notifie de cette conservation les Agents Contextes alors sélectionnés, même si ceux-ci ne se sont pas proposés au cycle actuel. Sinon, après avoir déterminé son action, mais avant de l'appliquer, l'Agent Contrôleur créé un nouvel Agent Contexte. Celui-ci est initialisé avec l'action précédemment choisie et mémorise la valeur courante des variables. Lors de son premier cycle de vie, il envoie une notifications aux Agents Variables et aux Agents Critères afin de signaler son existence et demander à recevoir leurs mises à jour. L'Agent Contrôleur va poursuivre cette même action tant que le niveau de criticité maximal diminue. Pendant ce temps, le nouvel Agent Contexte observe les variations de tous les niveaux de criticité pour définir ses prévisions. Enfin, lorsque l'action est abandonnée, l'Agent Contexte détermine ses plages de validité avec les minimums et maximums rencontrés pour chaque variable.

4.3.2 SNC 2 : Improductivité d'un Agent Contrôleur

Détection Lorsque, parmi toutes les propositions reçues, aucune action n'est associée à des prévisions de baisse du niveau de criticité, l'Agent Contrôleur est en SNC d'improductivité. Son processus de décision nominal (sélectionner l'action associée à la baisse la plus importante de criticité) ne produit aucune action. Deux résolutions sont possibles, selon les propositions reçues.

Résolution 1 Lorsque tous les types d'action (incrémenter, décrémenter, ne rien faire) sont proposés, l'Agent Contrôleur croit que le niveau de criticité maximal ne diminuera pas, quoi qu'il fasse. Il tente alors de faire diminuer le deuxième niveau de criticité le plus haut, (avec la contrainte de ne pas faire augmenter le premier). S'il n'y a toujours aucune proposition ne permettant de choisir une action, il s'intéresse au troisième niveau de criticité, et ainsi de suite récursivement. En dernier recours, il choisit le moindre mal : sélectionne l'action associée à la prévision de la plus petite hausse de niveau de criticité maximal.

Résolution 2 Dans le second cas, il existe des actions qui n'ont pas été testées dans la situation actuelle (puisque aucun Agent Contexte les représentant n'est valide). Les Agents Contextes actuellement valides indiquent en fait des actions à éviter, puisque provoquant selon eux une augmentation de la criticité. L'Agent Contrôleur va alors choisir parmi les actions possibles non proposées. Si sa précédente action a fait diminuer la criticité, il la conserve, sinon il l'inverse. L'Agent Contrôleur ne garde ce choix seulement dans le cas où la nouvelle action ainsi déterminée ne fait pas partie des actions à éviter. Sinon, il choisit aléatoirement la nouvelle action. Par exemple, si toutes les propositions sont d'incrémenter la valeur de l'entrée, l'Agent Contrôleur choisira au hasard entre décrémenter ou ne rien faire.

De manière analogue à la SNC 1, avant d'appliquer l'action choisie, l'Agent Contrôleur peut créer un nouvel Agent Contexte qui sera initialisé en suivant la même méthode.

4.3.3 SNC 3 : Conflit d'un Agent Contrôleur

Détection Lorsqu'un Agent Contrôleur applique une action qui avait été proposée par un Agent Contexte, il s'attend à ce que le niveau de criticité maximal évolue dans le sens indiqué par les prévisions annoncées. S'il s'aperçoit que ce n'est pas le cas, l'Agent Contrôleur a effectué une action potentiellement inadéquate. C'est pour lui une SNC de conflit.

Résolution Il ne faut pas continuer à appliquer cette action. L'Agent Contrôleur abandonne l'action en cours et en notifie les Agents Contextes qui l'avaient proposée lorsqu'elle a été sélectionnée. En outre, si l'Agent Contexte responsable des prévisions erronées est encore valide, il sera ignoré pour le prochain choix d'action.

4.3.4 SNC 4 : Conflit d'un Agent Contexte (prévisions fausses)

Détection Lorsque l'action d'un Agent Contexte valide est en train d'être appliquée, celui-ci observe les variations du niveau de criticité de tous les Agents Critères. Lorsque l'action est abandonnée, l'Agent Contexte compare ces variations avec ses propres prévisions. Il y a une SNC de conflit si l'observation d'au moins un niveau de criticité n'est pas en accord avec la prévision correspondante, c'est-à-dire si leur sens de variation sont opposés. En effet, l'Agent Contexte se rend alors compte qu'avoir proposé son action à l'Agent Contrôleur a induit ce dernier en erreur.

Résolution Une erreur dans le sens de variation d'une prévision n'est probablement pas une simple erreur d'observation initiale, ce n'est pas un problème d'ajustement de la prévision incriminée. Cela signifie que l'Agent Contexte n'aurait pas dû envoyer sa proposition (il n'aurait pas dû être valide à ce moment là). Il procède alors au rétrécissement de toutes ses plages de validité. Pour chacune d'entre elles, il rapproche la borne la plus proche vers la valeur courante de la variable.

4.3.5 SNC 5 : Conflit d'un Agent Contexte (prévisions inexactes)

Détection Similaire à la SNC 4, cette situation concerne également une différence, entre observations et prévisions, constatée par un Agent Contexte après avoir été relâché. Il s'agit ici du cas où le sens de chacune des prévisions correspond à celui des observations, mais où l'amplitude des variations est erronée. L'observation d'un tel phénomène est soumis au bruit sur les mesures présent dans la majorité des systèmes complexes. Aussi, une différence d'amplitude faible sera ignorée. Mais si celle-ci excède 5% (rappelons que le niveau de criticité varie entre 0 et 100), l'Agent Contexte est en SNC de conflit.

Résolution Un erreur dans l'amplitude d'une prévision est moins grave qu'une erreur dans le sens de celle-ci. Il s'agit simplement de l'ajuster au mieux. Aussi l'Agent Contexte ne modifie pas dans ce cas ses plages de validité, il se contente de diminuer ou d'augmenter ses prévisions erronées afin de les rapprocher de ses observations.

4.3.6 SNC 6 : Incompétence d'un Agent Contexte

Détection Il arrive qu'un Agent Contexte dont l'action est appliquée devienne non-valide mais ne reçoive pas de notification d'abandon de la part de l'Agent Contrôleur (par exemple dans le cas de la SNC 1). Il se trouve alors en SNC d'incompétence. Son action est en effet toujours en train d'être appliquée alors qu'il ne la préconise plus, ce qui est une situation non prévue par son comportement nominal.

Résolution Une solution pour l'Agent Contexte est d'étendre ses plages de validité afin de devenir valide. En effet de son point de vue, cette situation signifie que l'Agent Contrôleur a considéré que l'action qu'il avait proposée peut être prolongée au-delà de ses plages de validité actuelles. Il étend donc les bornes des plages qui provoquent son état de non-validité (autrement dit celles pour lesquelles la valeur courante de la variable est à l'extérieur des bornes).

4.3.7 SNC 7 : Inutilité d'un Agent Contexte

Détection Il peut arriver qu'un Agent Contexte soit amené à réduire progressivement une ou plusieurs de ses plages de validité, si bien que l'amplitude s'approche de zéro. C'est par exemple le cas lorsqu'un Agent Contexte se retrouve plusieurs fois dans la SNC 4, mais jamais dans la 6. Si une plage de validité atteint une amplitude inférieure à une taille critique (définie comme le centième de la plage de variation de la variable concernée), l'Agent Contexte considère que la probabilité d'être valide est trop faible et qu'il se trouve donc en SNC d'inutilité.

Résolution Un Agent Contexte inutile ne peut rien faire d'autre que se supprimer pour résoudre cette situation. Il évite ainsi d'occuper des ressources de calcul qui seraient profitables aux autres agents. Aussi, cette SNC n'est pas cruciale pour le bon fonctionnement d'ESCHER, la présence d'agents inutiles ne mettant pas en péril son adaptation. Elle permet cependant d'éviter un surplus du nombre d'agents.

4.3.8 SNC 8 : Improductivité d'un Agent Contexte (plages de validité)

Détection Cette SNC concerne un Agent Contexte qui a été valide, sélectionné, puis devenu non-valide (et donc relâché), et dont l'action a entraîné une baisse de criticité. C'est un cas idéal, et c'est pourquoi un Agent Contexte dans cette situation peut espérer que son action peut continuer à être pertinente, et penser pouvoir faire mieux que l'Agent Contexte par

lequel il a été remplacé. Il s'agit là d'une SNC d'improductivité : la décision de l'Agent Contexte de ne rien proposer (de ne pas être valide) est potentiellement inadéquate.

Résolution L'Agent Contexte étend les bornes des plages qui sont invalides. S'il a eu raison de faire cet ajustement, il sera probablement sélectionné plus longtemps la prochaine fois. S'il a eu tort, il tombera probablement dans la SNC 4 et rétablira la taille d'origine de ses plages. À l'image de la SNC 7, cette situation n'est pas capitale pour le bon fonctionnement d'ESCHER, mais permet d'affiner son apprentissage pour un risque limité.

4.3.9 SNC 9 : Improductivité d'un Agent Contexte (action proposée)

Détection Un Agent Contexte dont l'action a été sélectionné plusieurs fois consécutivement se considère en situation d'improductivité. En effet, il pense que, dans le cas idéal, son action devrait provoquer immédiatement les conséquences prévues sur les niveaux de criticité. Ses décision précédentes n'ont donc pas produit les bonnes actions à entreprendre. L'Agent Contexte va donc chercher à ajuster l'amplitude de l'action qu'il propose de manière à maximiser la diminution de criticité (ou minimiser son augmentation).

Résolution L'ajustement du pas d'action provient de l'estimation des effets de la variation de l'amplitude d'une action sur celle de la criticité. Le principe est d'augmenter ou diminuer le pas de manière à accélérer la diminution (ou à ralentir l'augmentation) de la criticité.

Pour cela, un Agent Contexte sélectionné plusieurs fois consécutivement va légèrement modifier aléatoirement l'amplitude de l'action qu'il propose et corréler ces variations avec celles de la vitesse de la criticité, qu'il observe. Ainsi, si la criticité est en train de diminuer :

- de plus en plus rapidement alors qu'il a augmenté le pas : l'Agent Contexte continue d'augmenter le pas ;
- de plus en plus rapidement alors qu'il a diminué le pas : l'Agent Contexte continue de diminuer le pas ;
- de moins en moins rapidement alors qu'il a augmenté le pas : l'Agent Contexte diminue le pas ;
- de moins en moins rapidement alors qu'il a diminué le pas : l'Agent Contexte augmente le pas.

L'Agent Contexte fait exactement l'opposé lorsque la criticité est en train d'augmenter, bien que ce cas se présente beaucoup moins fréquemment puisqu'il est rare que l'action d'un Agent Contexte soit conservée si elle a provoqué une augmentation de la criticité. Notons qu'une amplitude maximale de l'action peut être définie afin d'éviter des actions trop brusques.

La figure 4.7 illustre les effets de la résolution de cette SNC. Elle compare le contrôle d'une simple boîte noire SISO linéaire avec (à droite) et sans (à gauche) mécanisme d'ajustement du contrôle. La consigne initiale et de 5, une fois atteinte elle est placée à 2, puis de nouveau à 5, et ainsi de suite. À droite, on voit sur la première montée que la convergence vers la consigne est de plus en plus rapide (au contraire de la courbe de gauche où la vitesse est constante). De ce fait, la consigne est atteinte en environ 150 cycles de vie de l'Agent Contrôleur, contre environ

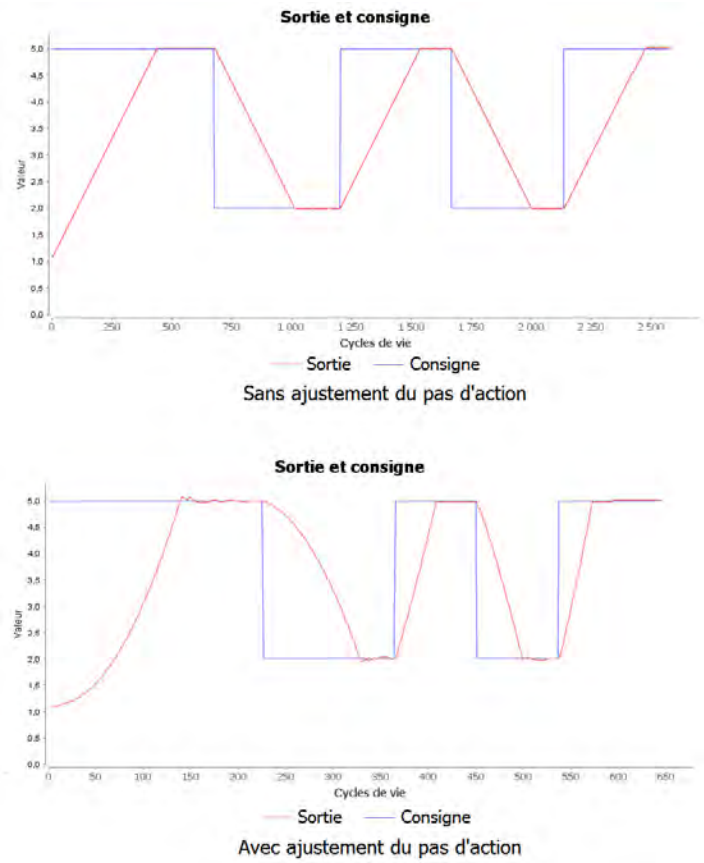

FIGURE 4.7 – Comparaison du contrôle sans et avec ajustement du pas d'action.

700 pour le test sans ajustement de l'amplitude de l'action. Cependant, cette accélération provoque un léger dépassement et quelques oscillations autour de la consigne (qui sont réduites petit à petit). Les effets sont similaires pendant la première descente. Pendant ces premières phases, les Agents Contextes sont en train d'ajuster l'amplitude. Lors des montées et descente suivantes, le système a appris que la bonne amplitude doit être forte au début (dans cet exemple, l'amplitude atteint la taille maximale fixée), puis faible lorsque la consigne est atteinte. Ainsi, la sortie arrive encore plus rapidement à la consigne (puisque la vitesse est maximale dès le début) et les oscillations sont beaucoup plus faibles, voire ont disparu (puisque l'amplitude de l'action des Agents Contextes valides autour de la consigne est faible).

4.3.10 Bilan des situations de non-coopération

Cette section a présenté les situations de non-coopération rencontrées par les agents d'ESCHER. Celles-ci provoquent la création, la suppression et l'ajustement des Agents Contextes, qui sont la mémoire du système. Autrement dit elles provoquent la mémorisation, l'oubli ou la correction d'informations à partir des observations sur le système réel : leur

résolution permet à ESCHER d'apprendre et de s'adapter.

En effet, les SNC 1 et 2 correspondent à l'acquisition de nouvelles information et apparaissent quand ESCHER découvre un système entièrement nouveau, ou bien un état encore inconnu d'un système déjà en partie exploré. Elles provoquent l'*ouverture* du système avec l'ajout de nouveaux Agents Contextes.

La SNC 3 permet à ESCHER de ne pas poursuivre une action en cas d'erreur, elle se résout grâce à la *réorganisation* des relations entre un Agent Contrôleur et certains de ses Agents Contextes. En effet, l'Agent Contrôleur abandonne l'action et, les cycles suivants, n'écoutera plus les Agents Contextes qui l'avaient proposée.

Chaque Agent Contexte s'évalue systématiquement, aussi les SNC 4 à 9 sont détectées lorsque l'un de ses élément n'est plus adapté au système contrôlé. Elles sont résolues par l'*ajustement* des agents (à l'exception de la SNC 7 qui est résolue par *ouverture*). Ainsi, ESCHER est en permanence en train de s'auto-évaluer et de s'adapter à son environnement.

Il y a deux scénarios typiques décrivant la vie possible d'un Agent Contexte. Le premier correspond au cas où son action est adéquate. Une fois créé, celui-ci est conservé, il étend assez fortement ses plages avant de se stabiliser. Si le système contrôlé évolue, il ajuste ses prévisions et réduit éventuellement ses plages. Si le système contrôlé change fortement, cela peut entraîner une importante régression, et donc la SNC 7 et la suppression de l'Agent Contexte.

Le second scénario correspond au cas où l'action initiale n'est pas adéquate. L'Agent Contexte est alors, le plus souvent, vite abandonné par l'Agent Contrôleur. Comme ses prévisions indiquent une augmentation de la criticité, il n'a que très peu de chances d'être à nouveau sélectionné. Cela n'est possible que dans le cas d'une SNC 2, ou bien si un autre Agent Contexte se retrouve valide au même moment, avec la même action, mais avec des prévisions inversées (ils peuvent dans ce cas être sélectionnés tous les deux simultanément). Cela veut dire que l'un des deux a tort, et celui-ci réduira ses plages de validité (SNC 4). Il peut s'agir de l'un comme de l'autre selon comment le système contrôlé a évolué. Dans tous les cas, l'un des deux finira probablement par se supprimer.

4.4 Consigne dynamique

Afin de ne pas en surcharger la présentation, un mécanisme a été jusqu'à maintenant mis de côté dans la description du système et des agents. Il s'agit de la consigne dynamique.

Que se passe-t-il si, une fois les Agents Contextes bien adaptés et les consignes atteintes, celles-ci sont changées par l'utilisateur ? Tel que le système a été présenté, les prévisions des Agents Contextes deviendraient toutes erronées, de nombreuses situations de non-coopération seraient détectées et le système devrait tout réapprendre pour satisfaire les nouvelles consignes. Ce n'est pas forcément très dérangeant la première fois, puisque ces nouvelles consignes pourraient impliquer l'exploration d'états du procédé encore inconnus. Mais dans le cas d'une consigne en créneau, par exemple, il est nécessaire de ne pas avoir à tout réapprendre.

La solution à ce problème consiste à représenter le paramétrage des Agents Critères grâce à des Agents Variables spécifiques. Par exemple, la valeur de la consigne à atteindre peut être représentée dans ESCHER par un Agent Variable. L'Agent Critère représentant la consigne se base alors sur cet agent et sur celui de l'entrée ou la sortie concernée du système contrôlé pour calculer son niveau de criticité. Ainsi, les Agents Contextes possèdent de nouvelles plages de validité, relatives aux Agent Variables des consignes. Leur apprentissage ne concerne alors plus uniquement le procédé contrôlé, mais aussi les souhaits de l'utilisateur. Intuitivement, il ne disent plus "lorsque le procédé est dans tel état, telle action a telles conséquences", mais "lorsque le procédé est dans tel état et que l'utilisateur a tels désirs, telle action a telles conséquences".

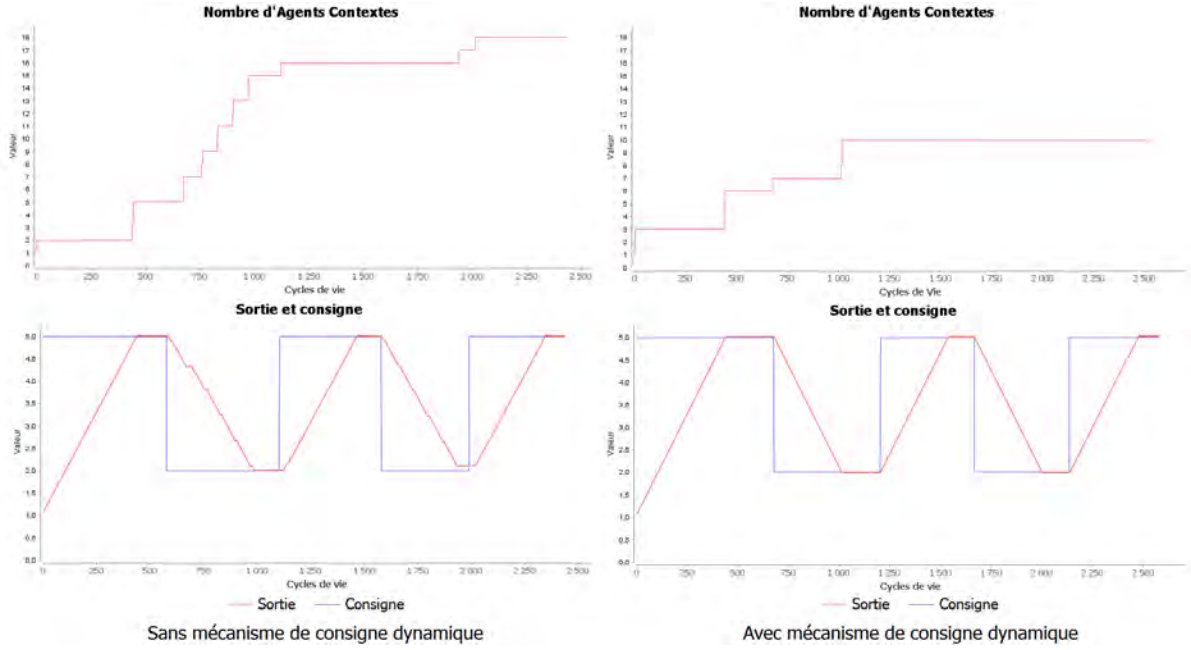

FIGURE 4.8 – Comparaison du contrôle sans et avec Agents Variables de consigne.

La figure 4.8 illustre l'effet de ce mécanisme sur le contrôle d'une simple boîte noire SISO linéaire pour laquelle on fait manuellement varier la consigne. On lance deux tests, l'un avec (à droite) et l'autre sans (à gauche) Agent Variable de consigne. Afin de mieux souligner l'impact de ce mécanisme, la SNC 9 a été désactivée pour cette illustration. On laisse ESCHER s'adapter et converger selon une même consigne initiale (la sortie doit atteindre 5). Une fois qu'il y parvient, on la modifie (la sortie doit maintenant atteindre 2), et on attend qu'il converge à nouveau. Puis on remet la consigne initiale, et ainsi de suite. Les courbes du haut montrent l'évolution du nombre d'Agents Contextes créés dans chaque cas, les courbes du bas montrent la consigne et la sortie de la boîte noire. On voit que sans ce mécanisme, passer de la première à la deuxième consigne et revenir est plus compliqué, de nombreux Agents Contextes sont créés. En outre, de nombreuses erreurs sont commises en cours de convergence (les paliers et légères remontées visibles lors des descentes par exemple), et la deuxième

consigne basse n'est même pas exactement atteinte. À droite cependant, la convergence est directe et il y a moins d'Agents Contextes créés.

4.5 Implémentation et instanciation

Cette section donne quelques détails sur l'implémentation d'ESCHER, et sur le paramétrage à effectuer pour une application dans un cas concret.

4.5.1 Ajustement des paramètres

Tous les paramètres auto-ajustés par les Agents Contextes le sont à l'aide de traqueurs de valeur adaptatifs (AVT, pour *Adaptive Value Trackers*, LEMOUZY, CAMPS et GLIZE 2011). Il s'agit des bornes des plages de validité, de l'amplitude de l'action, et des prévisions.

Un AVT converge vers une valeur à partir de *feedbacks* simples tels que "inférieur", "supérieur" et "égal". Il ajuste sa valeur et son pas de variation selon les *feedbacks* qu'il reçoit : le pas augmente si les *feedbacks* consécutifs sont identiques, diminue sinon. Ces variations du pas suivent des coefficients prédéfinis. La figure 4.9 montre l'évolution de la valeur d'un AVT paramétré de manière standard (deux mêmes *feedbacks* consécutifs doublent le pas, deux différents le divisent par trois).

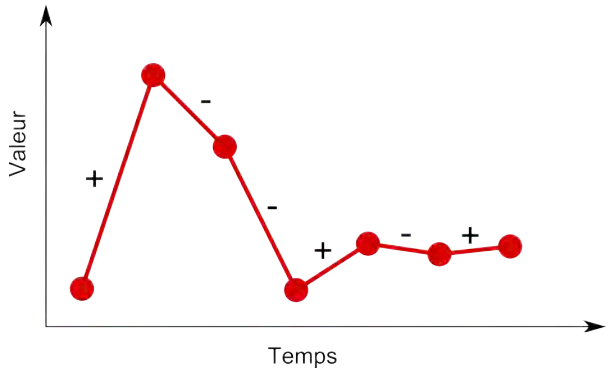

FIGURE 4.9 – Convergence d'un traqueur de valeur adaptatif.

Une partie des décisions d'un Agent Contexte traduit donc ses observations en *feedbacks* pour ses nombreux AVT. Par exemple, dans le cas de la SNC 5, s'il observe une variation de criticité plus importante que celle indiquée par sa prévision, un Agent Contexte envoie à l'AVT correspondant à cette dernière un *feedback* "supérieur". L'AVT augmente alors sa valeur d'une quantité égale à son pas courant. Bien sûr, la nouvelle prévision ne sera pas exactement égale à l'observation. Mais compte tenu de la nature dynamique de l'environnement et de la présence éventuelle de bruit, cette exactitude n'est pas nécessaire. Elle sera néanmoins atteinte si le système contrôlé est stable et que le même cas se présente plusieurs fois.

Les AVT convergent rapidement vers une valeur, sont capables de s'y stabiliser, et de repartir vers une nouvelle valeur tout aussi rapidement. Ils sont donc adéquats dans notre cas où les paramètres d'un agent peuvent fréquemment changer.

4.5.2 Algorithmes des comportements

L'activité de chaque agent lors d'un cycle de vie est présentée ici sous forme de pseudo-code.

4.5.2.1 Agents Variables

Algorithme 4.1 : Cycle de vie d'un Agent Variable

```
tant que Boîte de réception non vide faire
    Dépiler premier message;
    si Notification de nouvel agent alors
        Ajouter nouvel agent à la liste de destinataires;
    fin
fin
Acquérir la mesure;
Filtrer le bruit;
Envoyer la valeur à la liste de destinataires;
```

Les Agents Variables sont les plus simples du système. L'algorithme 4.1 présente leur comportement. Ils ajoutent d'éventuels destinataires, lisent la valeur de leur capteur, filtrent le bruit, et diffusent la valeur aux autres agents.

4.5.2.2 Agents Critères

Algorithme 4.2 : Cycle de vie d'un Agent Critère

```
tant que Boîte de réception non vide faire
    Dépiler premier message;
    si Notification de nouvel agent alors
        Ajouter nouvel agent à la liste de destinataires;
    fin
    si Mise à jour d'un Agent Variable alors
        Mettre à jour la valeur;
    fin
fin
Calculer le niveau de criticité;
Envoyer le niveau de criticité à la liste de destinataires;
```

Le comportement des Agents Critères, décrit par l'algorithme 4.2, est semblable à celui des Agents Variables. Ils mettent à jour leur liste de destinataire et la valeur des variables, puis calculent leur niveau de criticité et le diffusent.

4.5.2.3 Agents Contextes

Algorithme 4.3 : Cycle de vie d'un Agent Contexte

```
tant que Boîte de réception non vide faire
    Lire les messages (mises à jour valeur et criticité, notifications d'acceptation et de rejet);
    Mettre à jour représentations;
fin
si Vient d'être désélectionné alors
    Vérifier prévisions;
    si Prévisions incorrectes alors
        si Sens de variation opposé alors SNC 4
            Réduire plages de validité;
        sinon SNC 5
            Ajuster prévisions;
        fin
    fin
fin
si Sélectionné consécutivement deux fois ou plus alors SNC9
    Ajuster amplitude de l'action;
fin
si Non-valide et sélectionné alors SNC 6
    Étendre plages de validité;
fin
Vérifier et mettre à jour état de validité;
si Est devenu valide alors
    Envoyer proposition à l'Agent Contrôleur;
fin
si Est devenu non-valide alors
    Notifier l'Agent Contrôleur;
    si Était sélectionné et criticité a diminué alors SNC 8
        Étendre plages de validité;
    fin
fin
si Il existe une plage de validité de taille minimale alors SNC 7
    Se détruire;
fin
```

Les Agents Contextes sont les agents les plus propices à rencontrer des SNC. Ce sont eux qui ont en effet le plus de paramètres à ajuster. Ils se basent sur trois critères principaux pour détecter ces SNC : leur état de validité, leur état de sélection, et les variations de criticité. L'algorithme 4.3 détaille leurs décisions.

4.5.2.4 Agents Contrôleurs

Algorithme 4.4 : Cycle de vie d'un Agent Contrôleur

```
tant que Boîte de réception non vide faire
    Lire les messages (propositions d'Agents Contextes, notification de non-validité,
    mises à jour de valeurs et de criticité);
    Mettre à jour représentations (dont liste de propositions);
fin
si L'action appliquée au cycle de vie précédent n'a pas eu l'effet escompté alors SNC 3
    Envoyer une notification de rejet à l'Agent Contexte sélectionné;
fin
si Une action adéquate a été proposée alors
    Sélectionner l'action associée aux prévisions de diminution maximale de criticité;
    Envoyer une notification d'acceptation aux Agents Contextes qui l'ont proposée;
    Envoyer une notification de rejet aux Agents Contextes qui ont proposé une autre
    action;
sinon
    si Aucune action n'a été proposée alors SNC 1
        Décider seul d'une action;
        Créer un Agent Contexte;
    sinon
        si Toutes les propositions prévoient une augmentation de la criticité alors SNC 2
            si Toutes les actions possibles ont été proposées alors
                Sélectionner l'action associée aux prévisions minimales d'augmentation
                de criticité;
                Envoyer une notification d'acceptation aux Agents Contextes qui l'ont
                proposée;
                Envoyer une notification de rejet aux Agents Contextes qui ont proposé
                une autre action;
            sinon
                Décider seul d'une action parmi celles qui n'ont pas été proposées;
                Créer un Agent Contexte;
            fin
        fin
    fin
fin
Appliquer l'action choisie;
```

Les Agents Contrôleurs maintiennent une liste de propositions, contenant les actions et les prévisions de tous les Agents Contextes en cours de validité. C'est l'état de cette liste associé aux variations de criticités qui déterminent si un Agent Contrôleur est en SNC ou non. L'algorithme 4.4 montre le comportement de ces agents.

4.5.3 Architecture des agents

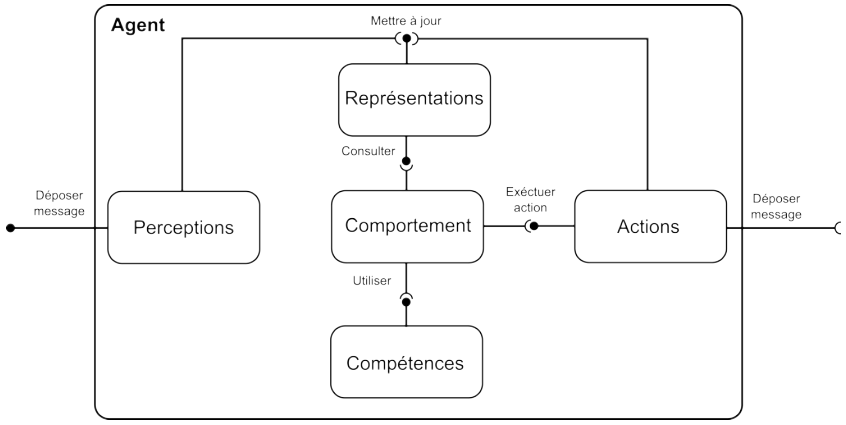

Figure 4.10 – Les principaux composants des agents d'ESCHER.

ESCHER a été implémenté à l'aide de MAY (*Make Agents Yourself*), un outil permettant de générer des squelettes de composants en JAVA à partir d'une description de l'architecture dans un langage spécifique (Noël, Arcangeli et Gleizes 2012). Les quatre types d'agents d'ESCHER ont globalement la même architecture de composants, à quelques exceptions près, ils ne diffèrent que par l'implémentation de ceux-ci. La figure 4.10 montre les principaux composants de cette architecture :

- Perception : contient les méthodes pour recevoir les messages, en extraire les informations utiles et les stocker dans le composant Représentations. Celui des Agents Variables récupère, en plus des messages, la valeur de la variable correspondante du procédé.
- Représentations : contient les connaissances de l'agent, qu'elles soient innées (présentes à l'initialisation) ou acquises grâce à la perception. Il s'agit par exemple de la liste des destinataires pour un Agent Variable.
- Compétences : contient des fonctions utiles au raisonnement de l'agent (par exemple vérifier sa validité pour un Agent Contexte)
- Comportement : contient les règles de décisions, faisant intervenir les représentations et les compétences de l'agent afin de décider d'actions à entreprendre.
- Actions : contient les méthodes pour exécuter des actions (c'est-à-dire l'envoi de messages, l'application d'une valeur en sortie pour les Agents Contrôleur, ou la modification de paramètres internes).

Ces composants sont parfois des composites, réutilisant des composants génériques (par exemple une boîte de réception de messages). D'autres composants, relatifs à l'aspect opératoire du système (gestion des processus, de l'interface graphique, etc), ont été implémentés mais ne sont pas intéressants ici.

4.5.4 Application à un cas concret

ESCHER se veut facile à instancier à un cas concret. Pour cela le nombre de paramètres à régler doit être réduit et le paramétrage ne doit pas nécessiter la mise en œuvre de techniques de calibration.

Les paramètres liés au procédé contrôlé à renseigner obligatoirement lors de l'initialisation du système sont :

- le nombre de variables contrôlables, et la référence de chacune ;
- le nombre de variables observables, et la référence de chacune.

Il est possible de renseigner les bornes minimales et maximales de chaque variable. Le système fonctionne sans cette connaissance, mais sa disponibilité peut faciliter la suite de l'instanciation, notamment la définition des souhaits de l'utilisateur, et peut servir de sécurité afin que ESCHER n'explore pas des zones de fonctionnement que l'on sait dangereuses pour le procédé. Dans tous les cas, ces derniers paramètres relèvent de connaissances basiques sur le système contrôlé et ne posent aucun problème.

Le point le plus délicat est, en fait, de définir les fonctions de criticité. Disposer des bornes aux variables permet d'utiliser une fonction appelée *fonction barrière* (voir annexe), qui est facilement paramétrable pour correspondre à des fonctions de consigne, de seuil ou même d'optimisation (si le domaine de définition est borné, il n'y a plus besoin d'asymptote pour ces dernières). Sans cela, il faut utiliser l'exponentielle dans la définition des fonctions de criticité, ce qui engendre un coût de calcul plus important. Les Agents Contrôleurs privilégient l'Agent Critère le plus critique. Cela signifie que la définition des compromis à atteindre se fait par le calage des fonctions de criticité. Par exemple, si l'on veut, de manière caricaturale, maximiser et minimiser la même variable, ESCHER placera cette dernière à la valeur pour laquelle les courbes de criticité se croisent. Un outil permettant de visualiser et modifier dynamiquement ces courbes permettrait une définition plus intuitive de ces fonctions, mais n'est pour le moment pas disponible. Cette connaissance ne concerne pas directement le procédé à contrôler, mais bien les objectifs de l'utilisateur. Ce dernier a toutefois besoin d'un minimum de connaissances sur le procédé s'il veut spécifier des objectifs réalistes. Ce sont des fonctions barrières qui ont été utilisées dans cette thèse. L'utilisateur n'a qu'a sélectionner le type de fonction (seuil, consigne ou optimisation), et préciser une valeur (pour un seuil ou une consigne), ou un sens (pour un seuil ou une optimisation).

Enfin, un certain nombre de paramètres sont secondaires, ils n'impactent pas significativement le comportement d'ESCHER, et n'ont donc pas à faire l'objet d'un réglage spécifique. Il s'agit par exemple de la taille minimale des plages de validité (qui va influer sur la conservations des Agents Contextes, il faut simplement faire attention qu'elle ne soit pas trop grande), de l'amplitude maximale d'une action (paramètres optionnel visant à empêcher ESCHER de faire des actions trop brusques, pour des raisons de sécurité), ou encore des paramètres des AVT (le pas minimal en détermine la précision et les coefficients influent sur la vitesse de convergence). La forte capacité d'adaptation des agents fait en sorte que ces paramètres internes n'ont pas une grande influence sur leur aptitude à trouver la meilleure organisation. Le tableau 4.1 résume tous ces paramètres et leur importance respective.

Table 4.1 – Paramètres de ESCHER.

Paramètres	Importance
Nombre de variables contrôlables	Obligatoire
Nombre de variables observables	Obligatoire
Référence des variables	Obligatoire
Plages de variations	Optionnel
Fonctions de criticités	Obligatoire
Amplitude maximale d'une action	Peu important
Taille minimale des plages de validité	Peu important
Pas minimal des AVT	Peu important
Coefficients des AVT	Peu important

Le chapitre 6 revient sur l'application d'ESCHER à un cas réel. Les sections suivantes prennent un peu de recul sur le système et analysent ses principales caractéristiques.

4.6 Un système de contrôle et d'apprentissage

ESCHER a été présenté comme un système de contrôle car il a été conçu pour cela. Néanmoins, l'apprentissage y joue un rôle de premier plan. Cette section approfondit ces deux aspects complémentaires du système, et cherche leurs liens.

4.6.1 ESCHER est un système de contrôle

La tâche première d'ESCHER est de contrôler un système inconnu (ou du moins dont on ne connaît que les entrées et les sorties). Les Agents Contrôleurs sont responsables de la valeur définie sur leur entrée. Nous exprimons ici de manière un peu plus formelle l'activité de ce type d'agent.

Si on note u_t la valeur courante de la variable d'entrée et a_t l'action appliquée au cycle de vie t par l'Agent Contrôleur correspondant, la prochaine valeur de la variable d'entrée s'exprime comme :

$$u_{t+1} = u_t + a_t$$

L'Agent Contrôleur choisit a_t à chaque cycle de vie t en fonction de ses représentations, qui sont composées de :

- $\mathcal{C}_t$, l'ensemble des niveaux de criticités des Agents Critères, mis à jour au cycle de vie t.
- $\mathcal{P}_t$, l'ensemble des propositions des Agents Contextes valides au cycle de vie t.

La proposition d'un Agent Contexte m valide au cycle de vie t de l'Agent Contrôleur est notée :

$$p_t^m := (a_t^m, F_t^m) \in \mathcal{P}_t$$

où a_t^m est une action et F_t^m est un ensemble de fonctions de prévision de niveaux de criticité. Ainsi une fonction $f_t^{m,i} \in F_t^m$, retourne le niveau de criticité de l'Agent Critère i prévu par

l'Agent Contexe m suite à l'application de l'action a_t^m :

$$f_t^{m,i}(a_t^m) = c_t^i + \delta^m(a_t^m) \tag{4.1}$$

où $c_t^i \in \mathcal{C}_t$ est la criticité de l'Agent Critère i perçue au cycle de vie t, et δ^m est une fonction résultant de l'apprentissage de l'Agent Contexte m. En pratique, un Agent Contexte m envoie une proposition d'action a_t^m accompagnée de l'ensemble des valeurs $f_t^{m,i}(a_t^m)$, et non un ensemble de fonctions $f_t^{m,i}$ calculables. L'expression 4.1 avait seulement pour but de faire apparaître explicitement une partie de l'apprentissage des Agents Contextes, dont nous reparlerons plus tard.

Pour chaque proposition p_t^m, on définit la fonction f_{max}^m comme celle qui retourne le plus haut niveau de criticité (en d'autres termes celle qui correspond à l'Agent Critère le plus critique) :

$$f_{max}^m := f_t^m \in F_t^m, f_t^m(a_t^m) = \max_{f \in F_t^m}(f(a_t^m))$$

C'est sur les valeurs des f_{max}^m qu'il reçoit que l'Agent Contrôleur base son choix, du moins dans le cas nominal.

4.6.1.1 Cas nominal

Dans le cas nominal, l'action a_t est choisie comme celle de la proposition dont la prévision de niveau de criticité est la plus basse tout en étant inférieure au niveau de criticité maximum courant :

$$a_t := a^i \in \mathcal{A}_t, f_{max}^i(a^i) = \min_m(f_{max}^m(a^m)) \wedge f_{max}^i(a^i) \leq \max \mathcal{C}_t \tag{4.2}$$

où $\mathcal{A}_t$ est l'ensemble des actions des propositions valides au cycle de vie t.

Le cas nominal est un cas idéal dans lequel l'Agent Contrôleur a reçu des propositions adéquates, c'est-à-dire parmi lesquelles il existe au moins une action permettant de faire baisser la criticité. Malheureusement, il ne se trouve pas toujours dans ce cas.

4.6.1.2 Situations de Non-Coopération

En effet, si :

$$\nexists f_{max}^i \in F_t^m, f_{max}^i(a^i) \leq \max \mathcal{C}_t \tag{4.3}$$

alors l'Agent Contrôleur ne se trouve pas dans le cas nominal mais dans une situation de non-coopération. L'équation 4.2 ne s'applique donc pas et le choix de a_t sera dicté par une partie de la résolution de la SNC. Il peut s'agir de la SNC 1 ou de la SNC 2.

SNC 1 Le premier cas se présente lorsque le procédé contrôlé se trouve dans une zone de son espace d'états encore non explorée par ESCHER. Il n'y a par conséquent aucun Agent Contexte valide, ce qui implique que $\mathcal{P}_t = \emptyset$, et donc $\mathcal{A}_t = \emptyset$. L'Agent Contrôleur va alors se

baser sur son action précédente et sur ce qu'il a observé des niveaux de criticité pour définir l'action à appliquer.

$$a_t := \begin{cases} a_{t-1} \text{ si } \max \mathcal{C}_t < \max \mathcal{C}_{t-1} \\ -a_{t-1} \text{ sinon} \end{cases}$$

Les deux autres cas surviennent lorsque l'Agent Contrôleur a bien une ou plusieurs propositions, mais qu'aucune n'est satisfaisante (c'est-à-dire qu'aucune ne prévoit une baisse du niveau de criticité maximum, voir condition 4.3). Il s'agit de la SNC 2. On note $\mathcal{A}$ l'ensemble de toutes les actions possibles pour l'Agent Contrôleur. On a donc à tout cycle de vie t : $\mathcal{A}_t \subseteq \mathcal{A}$.

SNC 2 - Cas 1 Si $\mathcal{A}_t = \mathcal{A}$, c'est que tout a déjà été tenté. On ne peut pas diminuer le niveau de criticité maximum. Le processus de décision de l'action est alors relancé, en se basant cette fois-ci sur le deuxième Agent Critère le plus critique et en ajoutant la contrainte de ne pas dégrader la criticité maximum. Si aucune décision n'est prise, on continue, itérativement, de passer à l'Agent Critère immédiatement moins critique. Si aucune action n'est choisie après la dernière itération, il n'y a rien de mieux à faire que de sélectionner l'action qui fait le moins augmenter le niveau de criticité maximum.

$$a_t := a^i \in \mathcal{A}_t, f^i_{max}(a^i) = \min_m(f^m_{max}(a^m))$$

SNC 2 - Cas 2 Enfin, lorsque $\mathcal{A}_t \neq \emptyset \wedge \mathcal{A}_t \neq \mathcal{A}$, cela signifie que certaines actions n'ont pas encore été essayées dans l'état courant du procédé. On note $\mathcal{A}_c = \mathcal{A} - \mathcal{A}_t$ l'ensemble de ces actions candidates, c'est-à-dire celles qui ne sont pas en train d'être proposées. L'Agent Contrôleur suppose que parmi ces actions s'en trouve une convenable, il va expérimenter en tirant aléatoirement une de ces actions dans $\mathcal{A}_c$.

$$a_t := \text{rand } (A_c)$$

La loi de probabilité sur $\mathcal{A}_c$ est uniforme dans le cas général. Cependant il existe des cas particuliers, qui ressemblent à la SNC 1 et pour lesquels une action prévaut par rapport aux autres :

$$\begin{cases} P(a_t = a_{t-1} | a_{t-1} \in \mathcal{A}_c \wedge \max \mathcal{C}_t < \max \mathcal{C}_{t-1}) = 1 \\ P(a_t = -a_{t-1} | -a_{t-1} \in \mathcal{A}_c \wedge \max \mathcal{C}_t > \max \mathcal{C}_{t-1}) = 1 \end{cases}$$

La formalisation présentée ici concerne uniquement l'activité de contrôle d'un Agent Contrôleur. Les aspects de couplage avec son groupe d'Agents Contextes, auxquels il envoie des *feedbacks* ont été négligés car ils concernent la deuxième facette de ESCHER.

De manière générale, la résolution des SNC ne se réduit pas au choix d'une action "de secours" à appliquer, elle provoque également la création, l'ajustement, et la suppression des Agents Contextes. Cela permet aux agents d'être de plus en plus adaptés, et donc au système de tendre vers l'adéquation fonctionnelle.

4.6.2 ESCHER est un système d'apprentissage

Le contrôle qu'effectue ESCHER sur un procédé s'améliore avec l'expérience, à mesure que les Agents Contrôleurs créent des Agents Contextes et que ces derniers s'ajustent. Il correspond donc à la caractérisation de l'apprentissage donnée dans la section 2.2 : c'est un programme dont la performance devient meilleure grâce à l'exploitation de données en cours de fonctionnement. Son protocole d'interaction avec l'environnement le place clairement dans la famille de l'apprentissage par renforcement.

Les ajustement engendrés par la résolution de SNC se déroulent au sein des Agents Contextes, qui sont responsables du stockage, de la maintenance et d'une partie de l'exploitation des données acquises en cours de vie du système. En effet, après avoir choisi l'action à appliquer, l'Agent Contrôleur envoie un *feedback* aux Agents Contextes qui l'avaient proposée. Ceux-ci utilisent ensuite cette information, couplée à leur observation des niveaux de criticité, pour déterminer s'ils doivent s'ajuster, et si oui, comment ils doivent le faire. Dans le cas où l'action n'avait été proposée par aucun Agent Contexte, l'Agent Contrôleur en crée un nouveau qui devra s'initialiser en accord avec ce qu'il percevra du procédé contrôlé. Aussi, on peut voir de l'apprentissage à deux niveaux : celui d'un seul Agent Contexte, et celui de l'ensemble des Agents Contextes associés à un Agent Contrôleur donné. Cet apprentissage aboutit à la coordination des Agents Contrôleurs, qui trouvent chacun la meilleure action à appliquer localement en fonction de l'état des autres.

4.6.2.1 Au niveau d'un Agent Contexte

La fonction d'un Agent Contexte est de donner une information fiable sur les effets d'une action sur l'environnement. Cette fiabilité concerne aussi bien le contenu de l'information que le moment où celle-ci est délivrée. Pour apprendre sa fonction, un Agent Contexte doit donc apprendre les prévisions, liées à l'action qu'il ajuste en parallèle, tout en trouvant sa place dans l'espace d'états de l'environnement (apprendre ses plages de validité, qui déterminent quand envoyer l'information). Cet apprentissage est déclenché lorsque l'Agent Contexte fait une proposition qui est acceptée.

Les prévisions dépendent de l'action que l'agent propose. Pour chaque niveau de criticité observé, il veut en déterminer la valeur future, et cherche donc la fonction f, telle que :

$$f(a) = c_t + \delta(a) = c_{t+k}$$

où a est l'action, c_t la valeur courante du niveau de criticité considéré (qui est perçu), et δ la variation apprise (à l'aide d'un AVT et de la SNC 5). Le nombre k dépend des plages de validité. En effet, la prévision s'effectue sur la durée totale de sélection de l'action, qui, dans le cas nominal, correspond à celle pendant laquelle l'Agent Contexte est valide.

Les plages de validité sont apprises de manière similaire aux prévisions, à l'aide d'AVT (un pour chaque borne) et des SNC 4, 6 et 8.

Chaque Agent Contexte apprend donc localement son action et ses prévisions, qui concernent la zone de l'espace d'états du système contrôlé qu'il occupe.

4.6.2.2 Au niveau d'un groupe d'Agents Contextes

L'ensemble des Agents Contextes associés à un Agent Contrôleur donné est une sorte de pavage (généralement partiel) de l'espace d'états de l'environnement. Chaque "tuile" est une zone pour laquelle les effets d'une action particulière sont connus. Les SNC 1, 2 et 7 conduisent à l'ajout ou à la suppression de "tuiles". Chacune d'elle est autonome : elle apprend elle-même sa taille et sa place, ainsi que l'action et les prévisions, et peut décider de disparaître. À la différence d'un véritable pavage, les Agents Contextes peuvent ici se recouvrir, partiellement ou complètement.

L'apprentissage de cet ensemble d'Agents Contextes concerne la mise en relation des états de l'environnement avec une action particulière et ses effets. L'ajustement des plages de validité est en fait une gestion du compromis entre généralisation et fidélité aux données. En effet, plus les plages d'un Agent Contexte sont larges, moins celui-ci tendra a être précis, il peut recouvrir des zones pour lesquelles des variations plus fines pourraient être observées. Par contre, il généralise la connaissance de l'effet d'une action à une zone étendue.

4.6.3 Comparaison avec des approches existantes

ESCHER adopte une approche locale de l'apprentissage et du contrôle, en rupture avec les techniques habituelles présentées dans le chapitre 2. L'autonomie des agents rend difficile une analyse globale du système. S'il est relativement aisé de décrire comment fonctionne un agent, faire le lien avec le comportement global du système est bien plus difficile. Cette section compare ESCHER avec des techniques existantes de contrôle et d'apprentissage afin d'en faciliter la compréhension.

4.6.3.1 Avec le contrôle dual

Dans le contrôle dual, le système contrôlé est inconnu (ou partiellement connu), et le contrôleur applique des actions soit pour en apprendre les conséquences sur les sorties (actions sondes), soit pour amener le système vers l'état attendu (actions de contrôle, voir section 2.1.2.4). Trouver l'équilibre entre ces deux types d'actions revient à résoudre une équation complexe, l'équation de Bellman (équation 2.2), ce qui est infaisable dans des cas réels. L'enjeu est de ne pas faire trop d'actions sondes afin de ne pas ralentir le contrôle, mais de ne pas non plus précipiter le contrôle afin de ne pas prendre le risque de dégrader le procédé.

ESCHER est également confronté à des systèmes inconnus et il apprend de ses actions. Cependant, contrairement au contrôle dual, il apprend de toutes ses actions, et toutes ses actions visent également à amener le système contrôlé vers l'état souhaité. En outre, il ne se base pas sur un modèle que l'apprentissage paramètre, mais sur les seules informations qu'il extrait de ses observations du système contrôlé.

La contrainte de devoir faire baisser la criticité (même lorsque aucun agent n'indique comment le faire) combiné au fait d'apprendre à partir de toute action peut être vu comme une approche de solution au problème de l'équilibre entre action de contrôle et action sonde.

Le contrôle guide l'apprentissage vers les états intéressants, se rapprochant de la consigne et empêchant de dévier vers des états éloignés.

4.6.3.2 Avec les systèmes de classeurs

Un système de classeurs (LCS, voir section 2.2.3.3) est un système d'apprentissage par renforcement. Il est composé d'une base de règles de comportement, d'un système d'appariement mettant en relation l'état de l'environnement avec les conditions des règles, d'un mécanisme de sélection parmi les différentes règles déclenchées, et enfin d'un algorithme génétique mettant à jour la base de règles.

Il existe des ressemblances entre un LCS et un Agent Contrôleur accompagné de son groupe d'Agents Contextes. Les Agents Contextes remplissent entre autres la fonction du système d'appariement (avec les plages de validité) et de la base de règles (en proposant des actions). L'Agent Contrôleur joue, quant à lui, un rôle similaire à celui du mécanisme de sélection d'action, puisqu'il doit choisir parmi plusieurs propositions.

La différence principale vient du fait que les Agents Contextes apprennent par eux-mêmes, en autonomie. Les règles d'un LCS sont au contraire évaluées par un algorithme génétique, qui élimine les plus faibles et en génère d'autres, normalement plus adaptées. La fonction d'évaluation de cet algorithme est un signal de récompense, perçu depuis l'environnement. Une difficulté dans l'instanciation d'un LCS est de répartir correctement ce signal parmi les règles afin de leur attribuer une mesure de force sur laquelle peut se baser l'algorithme génétique. Cette difficulté n'existe pas dans ESCHER, justement grâce à l'autonomie des agents. Ils évaluent eux-même leur pertinence et s'ajustent éventuellement. On peut cependant noter une similitude entre le signal de renforcement et les niveaux de criticité, sur lesquels se base l'apprentissage. En s'ajustant, les Agents Contextes proposent des actions de plus en plus adaptées, à des moments de plus en plus adéquats, et associées à des prévisions de plus en plus fiables. Ainsi, l'apprentissage nourrit le contrôle.

4.6.4 Le dilemme exploration-exploitation

Le dilemme entre action sonde et action de contrôle est le pendant, pour le contrôle, du dilemme exploration-exploitation de l'apprentissage. Dans ESCHER, les deux aspects contrôle et apprentissage sont fortement couplés : le contrôle nourrit l'apprentissage, qui en retour guide le contrôle (et inversement).

Ces dilemmes ne sont pourtant jamais apparus explicitement lors de la conception d'ESCHER. C'est là l'apport de l'approche locale des AMAS. En se focalisant sur le niveau local d'un agent, les problèmes du niveau global n'ont plus d'importance pour le concepteur. Ils seront réglés par les agents si ces derniers sont effectivement coopératifs. Dans notre cas, la solution du dilemme prend finalement la forme du couplage entre un Agent Contrôleur et un ensemble d'Agents Contextes pour chaque entrée contrôlée. Chaque Agent Contrôleur cherche à faire diminuer la criticité et chaque Agent Contexte cherche à être fiable. Le problème est résolu si chacun y parvient, ce que permet l'auto-organisation coopérative.

4.7 Un point sur l'auto-organisation dans ESCHER

Dans un AMAS, l'auto-organisation est motivée par la coopération. Elle est réalisée par la résolution de SNC, qui provoque des ajustements de paramètres internes aux agents, de leur relations, ou encore la création ou la suppression d'agents. Dans ESCHER, cette auto-organisation est principalement le fruit des Agents Contextes, mais pas uniquement.

4.7.1 Au niveau des Agents Contextes

À l'initialisation, ESCHER ne dispose d'aucun Agent Contexte. Ils sont tous créés au cours de l'exécution du système, et suivent ensuite de simples règles locales de comportement et d'ajustement. Ces règles mènent à la formation, pour chaque entrée contrôlée, d'un ensemble cohérent d'Agents Contextes dans lequel chacun représente une portion de l'espace d'états de l'environnement pour laquelle les effets d'une action sur les niveaux de criticité sont connus. Ces agents sont en permanence en train de s'adapter de manière autonome aux changements de l'environnement. On peut donc dire que les "sous-SMA" composés d'un groupe d'Agents Contextes et d'un Agent Contrôleur sont auto-organisateurs.

4.7.2 Au niveau des Agents Contrôleurs

Un autre point de vue fait apparaître de l'auto-organisation dans ESCHER, mais de manière moins explicite. Chaque Agent Contrôleur prend ses propres décisions quant à l'action à appliquer sur son effecteur. Ces décisions sont prises localement, il ne négocie jamais avec les autres Agents Contrôleurs. Pourtant, ils parviennent ensemble à diminuer les niveaux de criticité, amenant le système contrôlé dans l'état désiré.

Cela est dû au fait qu'ils partagent le même environnement, et notamment les mêmes Agents Variables, dont certains représentent les entrées du système contrôlé. Ainsi, les propositions des Agents Contextes pour un Agent Contrôleur donné sont en partie conditionnées par l'activité des autres Agents Contrôleurs. Le comportement d'un Agent Contrôleur est donc influencé par celui des autres, et inversement. Cela peut être vu comme une forme d'auto-organisation. Chacun décide localement, mais le comportement global du système (les actions sur toutes les entrées contrôlées) est cohérent.

4.7.3 De l'émergence dans ESCHER ?

On peut se poser la question de savoir si le produit de cette auto-organisation est émergent. Il y a plusieurs réponses à cette question. Tout d'abord, nous avons vu qu'il y a au moins deux manières de voir l'auto-organisation dans ESCHER.

Au niveau des Agents Contrôleurs, le résultat de l'auto-organisation est la coordination des actions qui provoquent la diminution des niveaux de criticité. Ce contrôle repose directement sur l'activité des groupes d'Agents Contextes. Si la configuration de ces derniers est émergente, alors on peut qualifier le contrôle d'émergent.

Au niveau des Agents Contextes, le résultat de l'auto-organisation est un recouvrement de l'espace d'états de l'environnement par les différentes plages de validité, et l'approximation des fonctions de criticité donnée par les prévisions. Comme aucun des Agents Contextes n'a d'information sur l'état global du groupe dont il fait partie, on peut considérer que la configuration de ce groupe a un moment donné est émergente. En outre, il est très difficile de prévoir quelle sera cette configuration longtemps en avance, même en connaissant parfaitement le comportement d'un Agent Contexte. Toutefois, cela est en grande partie du à un manque d'information sur l'environnement, et ne constitue peut-être pas un argument recevable pour parler d'émergence ici.

La section 3.2.2.1 appuie sur le fait que l'émergence apparait lorsque l'on ne sait pas faire complètement le lien entre le niveau micro (ici les agents) et le niveau macro (l'activité globale de contrôle dans notre cas, ainsi que, éventuellement, la configuration des groupes d'Agents Contextes). Les connaissances de l'observateur sont alors impliquées dans la qualification du comportement global comme émergent ou non. Or, ces connaissances sont sujettes au changement. Si le comportement global de ESCHER est un jour entièrement compris et formalisé, perdra-t-il son caractère émergent ?

Cette question laisse à penser qu'il est plus pertinent de parler d'auto-organisation que d'émergence dans le cadre de l'analyse et la conception de SMA.

4.8 ESCHER se base-t-il sur un modèle ?

Les contrôleurs ne se basant pas sur un modèle du procédé sont intéressants car ils ne nécessitent pas la lourde tâche de la construction et du paramétrage d'un modèle mathématique pour fonctionner. De tels contrôleurs sont dits *model-free*. Qu'en est-il de ESCHER ?

4.8.1 Non

La réponse évidente est : non. En effet, ESCHER ne bénéficie d'aucun modèle mathématique préétabli, paramétré ou non, du système contrôlé pour en calculer le comportement. Il apprend le contrôle directement à partir de ses observations sur les entrées et les sorties du procédé.

4.8.2 Oui

Un modèle est une représentation de la réalité. Il est une approximation qui permet de se concentrer sur les caractéristiques pertinentes en regard de l'utilisation pour laquelle il est construit.

L'ensemble des Agents Contextes associé à un Agent Contrôleur donné représente les conséquences des actions sur le procédé vis-à-vis des objectifs de l'utilisateur. En ce sens, il peut être considéré comme un modèle, non du système contrôlé, mais de son contrôle lui-même. Ce modèle n'est pas donné *a priori*, il est appris entièrement pendant le fonctionnement et est en constante évolution. En outre, il est peu probable qu'il soit complet, c'est-à-dire

que toutes les actions pour toutes les consignes à partir de tous les états possibles soient représentées. Mais cette complétude n'est pas nécessaire puisque les capacités d'adaptation du système le rendent capable de gérer des situations inconnues.

4.9 Résumé

Ce chapitre a présenté ESCHER, un système multi-agent capable d'apprendre le contrôle d'un procédé à partir de la seule observation de ses entrées et sorties.

Dans un premier temps, nous avons abordé le comportement nominal du système et des agents, celui qui permet de réaliser la fonction globale lorsqu'il est adapté. Nous avons d'abord introduit les quatre types d'agents en précisant leur fonction locale, puis nous avons présenté une vue générale du système. Un exemple simple a permis de montrer comment les connaissances distribuées dans les Agents Contextes sont utilisées pour contrôler un procédé dont on ne sait rien du fonctionnement interne.

Dans un deuxième temps, les mécanismes d'auto-organisation des agents ont été expliqués. Conformément à l'approche AMAS, ces derniers sont basés sur la détection et la résolution de situations de non-coopération. Ils permettent à ESCHER d'apprendre le contrôle en s'adaptant au système contrôlé ainsi qu'aux objectifs de l'utilisateur.

Enfin, l'implémentation et l'instanciation du système ont été abordées. Nous avons vu qu'ESCHER n'a besoin que de renseignements basiques sur le procédé, comme la connaissance des entrées et des sorties, pour pouvoir fonctionner. Nous avons ensuite pris un peu de recul sur ESCHER. Un point a été fait sur la complémentarité du contrôle et de l'apprentissage, un autre sur l'auto-organisation et l'émergence, et finalement un dernier sur l'utilisation ou non d'un modèle.

Il est maintenant temps de s'intéresser à ce que l'on obtient lorsque l'on applique ESCHER sur divers systèmes. Mais avant de présenter les expériences menées avec notre système, il nous faut aborder le problème de la disponibilité du procédé à contrôler. En effet, lors d'un projet, le système cible (ou un simulateur complet de celui-ci) n'est que rarement disponible pour tester et développer le contrôleur. Dans le cas d'une approche "boîte noire" comme la nôtre, le contrôleur peut être testé sur n'importe quel type de "simulateur", qu'il corresponde à un système réel ou non, du moment qu'il possède des caractéristiques de complexité et de dynamique similaires à celle du procédé cible. Aussi, avoir un générateur automatique de boîtes noires, permettant de disposer de différentes instances de systèmes à contrôler, peut être très utile.

Le prochain chapitre présente un tel générateur, conçu et développé durant cette thèse, et très utilisé pendant le développement d'ESCHER.

CHAPITRE 5

BACH, compositeur de boîtes noires

Ce chapitre présente un système multi-agent appelé BACH (*Builder of Abstract maCHines*). Ce système construit automatiquement une "boîte noire" à partir d'exigences renseignées par l'utilisateur. Il permet ainsi de produire diverses instances de tests pour des systèmes de contrôle capables d'apprentissage.

5.1 Les besoins

Généralement, les systèmes de contrôle sont d'abord testés sur des simulations, avant d'être appliqués au procédé réel. Ces simulations se basent sur des modèles précis du procédé ciblé, qui peuvent être très difficiles à obtenir. Toutefois, dans le cas d'une approche de type "boîte noire", comme la nôtre, la fidélité de la simulation au système réel n'est pas une nécessité. Autrement dit, le modèle utilisé par la simulation n'a pas besoin de reproduire précisément le comportement d'un système réel particulier, ni même d'en avoir la sémantique. Il doit reproduire un type de comportement (celui des systèmes complexes).

En effet, les connaissances du contrôleur sur le procédé sont acquises au cours de son fonctionnement. Une grande part de la difficulté d'une telle approche est donc liée au fait d'apprendre. Les tests concernent alors les capacités d'apprentissage et d'adaptation à un environnement complexe du contrôleur. Comme celui-ci n'a aucune connaissance du fonctionnement interne du système qu'il contrôle, il est possible de le tester sur n'importe quel type de procédé artificiel, même ceux dont les entrées et sorties ne correspondent à rien de concret. Il est néanmoins nécessaire que le comportement de la "boîte" contrôlée soit d'une complexité semblable à celle du procédé cible.

Cette complexité se manifeste de plusieurs manières lorsque l'on adopte une observation extérieure d'un système. Tout d'abord, l'interdépendance entre les entrées et les sorties peut poser des difficultés. Une entrée est susceptible d'influencer plusieurs sorties, tout comme une sortie peut être influencée par plusieurs entrées. Ensuite, ces influences sont bien souvent non-linéaires, on peut observer des phénomènes tels que des accélérations, des oscillations, ou

encore des amortissements. Enfin, le bruit et la latence compliquent encore la reconnaissance des effets d'une entrée sur une sortie donnée.

Ainsi, les tests en cours de développement d'un système de contrôle capable d'apprentissage visent à vérifier comment celui-ci se comporte sous diverses configurations. On veut étudier des cas bien précis et disposer de plusieurs instances de chaque cas. Il faut donc un modèle de "boîtes noires" sur lequel on peut modifier la dépendance entre les entrées et les sorties, faire varier leur nombre, ou encore maîtriser l'apparition d'oscillations, peu importe la signification des données vis-à-vis du procédé du monde réel (dans notre cas un moteur à combustion).

Disposer de telles *boîtes noires abstraites* (puisqu'elles ne correspondent à rien de concret), exhibant divers aspects de complexité, a permis de tester ESCHER durant son développement, et de facilement identifier et corriger les comportements inadéquats. Pouvoir générer automatiquement ces boîtes noires, d'après des exigences définies par l'utilisateur, a offert une grande souplesse au processus de développement et apporté une alternative à l'obtention d'un simulateur ou à de nombreux tests coûteux sur un véritable moteur. Ces derniers ont pu être réservés à la phase finale de validation. Les sections suivantes présentent ces boîtes noires abstraites ainsi que le système permettant de les générer.

5.2 Les boîtes noires abstraites

Une boîte noire abstraite est constituée d'entrées, de sorties et de fonctions (au sens mathématique du terme). Seules les entrées et les sorties sont visibles depuis l'extérieur. Chaque entrée est reliée à au moins une sortie par une chaîne de fonctions, et inversement. Les différentes chaînes de fonctions d'une boîte noire peuvent être entremêlées et présenter des cycles. Le rôle des entrées et sorties est de faire le lien entre les fonctions et l'extérieur. Une fonction dispose de ports d'entrée et d'un port de sortie. Elle lit ses arguments sur ses ports d'entrée et les utilise pour calculer sa valeur, qu'elle diffuse sur son port de sortie. Un port d'entrée est lié soit à une entrée de la boîte noire, soit à un port de sortie (de la

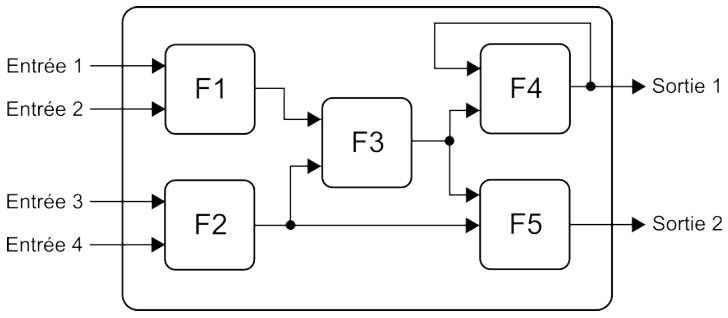

Figure 5.1 – Exemple de boîte noire abstraite.

même fonction, ou d'une autre). Un port de sortie est lié à une ou plusieurs sorties de la boîte ainsi qu'à un ou plusieurs ports d'entrée (de la même fonction, ou d'une autre). La

figure 5.1 montre un exemple d'une telle boîte, comprenant quatre entrées, deux sorties et cinq fonctions. Dans cet exemple, toutes les entrées influent sur toutes les sorties. On peut également voir un cycle (la sortie de la fonction F4 boucle sur sa propre entrée).

Afin de simplifier le processus de génération, et parce que cela n'a aucun impact d'un point de vue externe (c'est-à-dire du point de vue du contrôleur qui sera testé), le nombre de ports d'entrée d'une fonction est limité à deux.

5.3 La génération automatique

Générer une boîte noire abstraite consiste en fait à produire une instance du modèle présenté par la figure 5.2. Une fonction doit avoir chacun de ses ports d'entrée lié à un, et un

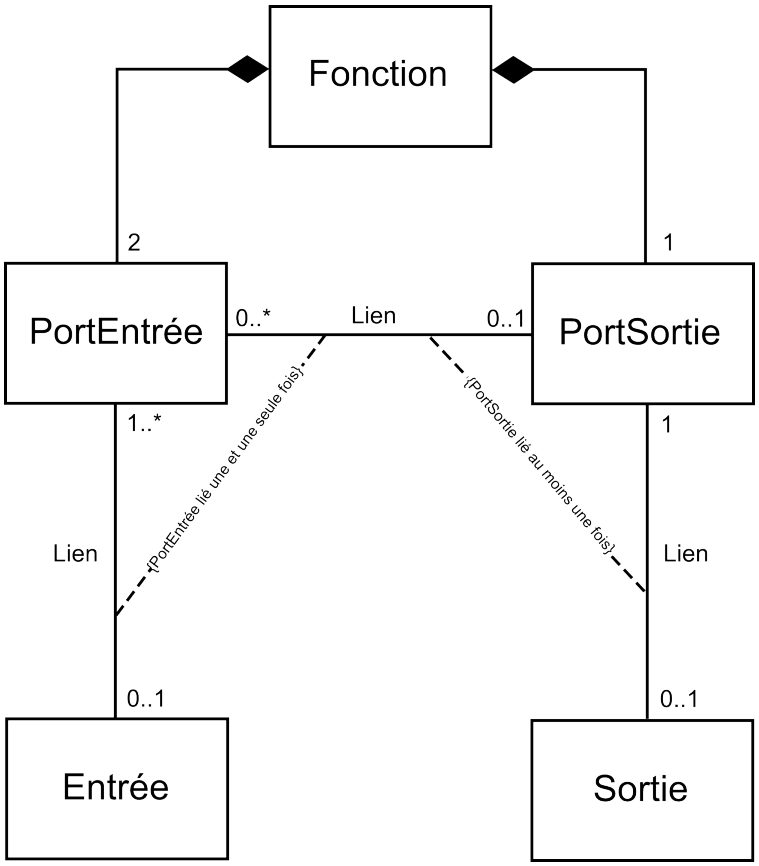

FIGURE 5.2 – Modèle des boîtes noires abstraites.

seul, partenaire qui peut être une entrée de la boîte ou le port de sortie d'une fonction. Les sorties de la boîte noire doivent chacune être liée à un seul port de sortie de fonction. Les entrées de la boîte doivent être liées à un ou plusieurs ports d'entrée de fonction. Enfin, le port de sortie d'une fonction doit être lié à un ou plusieurs partenaires, qui peuvent être des sorties de la boîte comme des ports d'entrée de fonction.

En plus d'être conforme à ce modèle, une boîte noire abstraite doit respecter des exigences définies par l'utilisateur. Celui-ci choisit :

- le nombre d'entrées, et de sorties ;
- le nombre de fonctions à l'intérieur de la boîte ;
- l'interdépendance des entrées et des sorties (il définit quelle entrée influence quelle sortie) ;
- le pourcentage de fonctions qui feront partie d'un cycle (cela influe notamment sur la présence d'oscillations sur les sorties de la boîte) ;
- les plages de variation des entrées et des sorties de la boîte.

L'approche AMAS a été utilisée pour concevoir BACH, un système qui génère des boîtes noires abstraites conformes au modèle de la figure 5.2, et respectant les exigences définies par l'utilisateur. Aussi, les entrées et les sorties d'une boîte noire, ainsi que les fonctions, sont transformées en agents. La génération se déroule en deux étapes :

- L'auto-assemblage, pendant laquelle les agents forment des liens, des entrées vers les sorties à travers des fonctions, tout en étant conforme au modèle et en respectant certaines des contraintes : l'interdépendance, le nombre de composants (entrées, sorties et fonctions), et la présence de cycles.
- L'auto-ajustement, pendant laquelle les fonctions ajustent leurs paramètres internes afin que les plages de variations des entrées et des sorties soient respectées.

Les sections suivantes présentent le comportement nominal des agents, puis les deux étapes de la génération.

5.3.1 Présentation générale

Cette section commence par une présentation des agents de BACH et de leur comportement nominal. Ce comportement correspond au cas où la boîte noire est construite, et que les agents n'ont qu'à effectuer la transformation des valeurs des entrées en valeurs de sortie. S'ensuit une brève explication des SNC, qui seront détaillées dans les sections suivantes. Dans BACH, chaque entrée est un agent, ainsi que chaque sortie, et chaque fonction.

5.3.1.1 Agent Entrée

Le comportement nominal d'un Agent Entrée est simplement de percevoir sa valeur depuis l'environnement et de la transmettre à tous les ports d'entrée d'Agents Fonctions auxquels il est lié. Cette valeur peut être définie par un utilisateur humain, ou par un autre programme.

5.3.1.2 Agent Sortie

Le comportement nominal d'un Agent Sortie est simplement de lire la valeur du port de sortie de l'Agent Fonction auquel il est lié. Cette information peut alors être affichée par une interface graphique ou transmise à un autre programme.

5.3.1.3 Agent Fonction

Le comportement nominal d'un Agent Fonction est de lire la valeur de chacun de ses deux ports d'entrées, puis d'appliquer sur ces valeurs la fonction mathématique qu'elle embarque afin de calculer la prochaine valeur à transmettre via son port de sortie. La valeur de chacun des ports d'entrée est donnée par l'agent qui y est lié. Il peut s'agir d'un Agent Entrée, ou d'un Agent Fonction (éventuellement soi-même). Le port de sortie d'un Agent Fonction peut être lié à plusieurs agents (un Agent Sortie, ou des Agents Fonctions, éventuellement soi-même). La fonction appliquées aux valeurs des ports d'entrée est une fonction mathématique quelconque à deux arguments. Ce sujet est détaillé dans la section 5.3.3.

5.3.1.4 Situations de non-coopération

Une situation de non-coopération correspond à un état dans lequel un agent est incapable de réaliser son comportement nominal.

Dans le cas de BACH, ces situations surviennent principalement lorsqu'un agent, quel que soit son type, n'a pas suffisamment de partenaires liés. En effet, pour fonctionner, un Agent Entrée doit avoir au moins un partenaire à qui transmettre sa valeur, un Agent Sortie doit avoir un et un seul partenaire lui transmettant une valeur, et un Agent Fonction doit avoir un partenaire lié sur chaque port d'entrée, et au moins un sur son port de sortie. En outre, les différents liens doivent produire une structure globale qui respecte les exigences de l'utilisateur. La phase d'auto-composition est celle durant laquelle les agents forment des liens sous certaines contraintes, afin de produire une structure adéquate.

Un deuxième type de SNC survient lorsque les plages de variation des Agent Sorties, qui font partie des exigences définies par l'utilisateur, ne sont pas respectées. C'est alors aux Agent Fonctions de s'ajuster. Cette SNC ne peut être détectée et résolue que si les liens ont été formés, c'est pourquoi la phase d'auto-ajustement vient après celle d'auto-composition.

5.3.2 L'auto-composition

BACH est initialisé avec autant d'agents que ce que l'utilisateur a spécifié. Déduites des exigences de l'utilisateur, des contraintes sont associées à chacun des agents. Ils peuvent les satisfaire en se liant avec d'autres agents. Elles sont de trois types :

- Les *contraintes de lien* : elles représentent la nécessité pour un agent d'avoir le nombre minimal requis de partenaires pour que la boîte noire soit conforme au modèle de la figure 5.2.
- Les *contraintes de chemin* : elles représentent, pour un agent, la nécessité d'être lié à une liste donnée d'Agents Entrées (directement, ou à travers des Agents Fonctions). Ces contraintes sont calculées à partir des exigences d'interdépendance.
- Les *contraintes de cycle* : elles expriment la nécessité pour un Agent Fonction de faire partie d'un cycle dans une chaine d'Agents Fonctions.

Un agent se considère en SNC tant que toutes ses contraintes ne sont pas résolues. Celles-ci sont associées à un niveau de criticité, qui permet de les hiérarchiser. Un agent traite en

priorité sa contrainte la plus critique, et le niveau de criticité total d'un agent est la somme de celui de chacune de ses contraintes.

Les paragraphes suivants décrivent, pour chaque type d'agent, les contraintes qui leur sont associées et le comportement suivi pour les résoudre.

Agents Entrées Un Agent Entrée n'a qu'un seul type de contrainte à résoudre. Il s'agit d'une contrainte de lien lui imposant de se lier à au moins un partenaire (dans ce cas un Agent Fonction). Le tableau 5.1 montre le comportement associé à la résolution de cette contrainte pour un Agent Entrée.

Table 5.1 – Contraintes d'un Agent Entrée.

Contrainte	Résolution
Contrainte de lien : L'agent doit se lier à au moins un port d'entrée d'un Agent Fonction. Niveau de criticité : 1.	Chercher un Agent Fonction avec un port d'entrée libre, et s'y lier.

Agents Sorties Un Agent sortie est initialisé avec deux types de contraintes : une contrainte de lien, afin qu'il se lie avec un Agent Fonction, et une contrainte de chemin, afin que les entrées spécifiées par l'utilisateur influencent sa valeur. Le tableau 5.2 expose ces contraintes,

Table 5.2 – Contraintes d'un Agent Sortie.

Contrainte	Résolution
Contrainte de lien : L'agent doit se lier avec un, et un seul, port de sortie d'un Agent Fonction. Niveau de criticité : 2.	Chercher un Agent Fonction et s'y lier.
Contrainte de chemin : L'agent doit être relié, à travers une chaîne d'Agents Fonctions, à tous les Agents Entrées d'une liste donnée. Cette liste est déduite des exigences de l'utilisateur. Niveau de criticité : 2*taille de la liste.	Chercher un Agent Fonction, si possible déjà relié à tout ou partie de la liste (et à aucun autre Agent Entrée). Se lier à cet Agent Fonction et lui transmettre cette contrainte de chemin.

ainsi que le comportement adopté par un Agent Sortie pour les résoudre. Une contrainte de chemin est considérée comme résolue une fois que celle-ci a été délégué à un partenaire.

Agents Fonctions Un Agent Fonction est initialisé avec une contrainte de lien, et éventuellement une contrainte de cycle (l'utilisateur indique la proportion d'Agents Fonctions à recevoir une contrainte de cycle). En outre, un Agent Fonction peut recevoir une contrainte de chemin,

déléguée par un partenaire lié à son port de sortie. Les Agents Fonctions sont les agents les plus sollicités durant l'auto-composition, aussi le niveau de criticité de leurs contraintes est plus élevé que celui des autres agents. Le tableau 5.3 montre ces contraintes, et leur résolution.

Table 5.3 – Contraintes d'un Agent Fonction.

Contrainte	Résolution
Contrainte de lien : L'agent doit lier son port de sortie à au moins un agent. Niveau de criticité : 3.	Chercher un partenaire libre, et s'y lier. Un partenaire peut être un Agent Sortie, ou un autre Agent Fonction (le lien se fait avec un des ports d'entrée du partenaire).
Contrainte de lien : L'agent doit lier chacun de ses ports d'entrée à un seul agent. Niveau de criticité : 2 (1 pour chaque port).	Chercher un partenaire pour chaque port, et s'y lier. Un partenaire peut être un Agent Entrée ou un autre Agent Fonction (le lien se fait avec le port de sortie du partenaire). Le même partenaire peut être lié aux deux ports d'entrée.
Contrainte de chemin : L'agent doit être relié à tous les Agents Entrées de la liste spécifiée, soit directement, soit via d'autres Agents Fonctions. Les Agents Fonctions ne sont pas initialisés avec cette contrainte, elle est reçue de partenaires liés au port de sortie. Niveau de criticité : 4*taille de la liste.	Lorsque l'agent reçoit cette contrainte, il regarde les Agents Entrées auxquels il est déjà relié, et les retire de la liste. Ensuite, si la taille de la liste est inférieure ou égale au nombre de ports d'entrée libres de l'agent, alors il les lie directement aux Agents Entrées spécifiés, et la contrainte est résolue. Sinon, la liste est divisée en deux parties de taille équivalente, et chacune est associée à un port d'entrée. Si ce port est déjà lié à un Agent Fonction, la demi-liste lui est déléguée sous la forme d'une nouvelle contrainte de chemin. Sinon, l'agent cherche un Agent Fonction (si possible déjà relié aux Agents Entrées de la contrainte), s'y lie et lui délègue la demi-liste.
Contrainte de cycle : L'agent doit faire partie d'un cycle dans une chaîne d'Agents Fonction. Niveau de criticité : 7	L'agent choisit, selon la disponibilité de partenaires, entre : lier son port de sortie à un Agent Fonction présent dans la chaîne menant à un port d'entrée, ou lier un port d'entrée à un des Agents Fonctions de la chaîne partant de son port de sortie, ou encore lier son port de sortie à l'un de ses propres ports d'entrée.

Le niveau de criticité des contraintes a été choisi de manière à privilégier les plus difficiles à résoudre.

Le choix d'un partenaire Plusieurs des comportements de résolution de contraintes impliquent la recherche (et donc le choix) d'un partenaire pour s'y lier. Ce choix est basé sur deux critères principaux : l'utilité du partenaire potentiel, et son niveau de criticité.

Les partenaires potentiels sont d'abord triés en deux catégories, selon qu'ils puissent répondre aux besoins de l'agent demandeur ou non. Par exemple, dans le cas d'une contrainte de chemin, tous les Agents Fonctions déjà reliés à des Agents Entrées qui ne font pas partie de la contrainte sont éliminés. Parmi les partenaires potentiels restants, l'agent choisit celui pour lequel le lien serait le plus bénéfique, c'est-à-dire dont le niveau de criticité diminuerait le plus.

Pendant le processus d'auto-composition, il peut arriver qu'un agent se trouve sans partenaire potentiel adéquat. Cela survient lorsque l'utilisateur a posé un problème surcontraint, et la solution consiste donc à enfreindre une exigence. Étant donné l'utilisation future de la boîte noire, l'exigence la moins importante est celle concernant le nombre de fonctions dans la boîte. C'est elle qui est alors relâchée. Un nouvel Agent Fonction est donc créé pour servir de nouveau partenaire.

La structure qui résulte de ce processus d'auto-composition assure à la boîte noire d'être conforme au modèle et de respecter toutes les exigences de l'utilisateur, excepté celles concernant les plages de variations des entrées et des sorties. C'est à ce moment que les Agents Fonctions peuvent s'ajuster afin que ces plages soient respectées. La section suivante explique comment se déroule cet auto-ajustement.

5.3.3 L'auto-ajustement

Les Agents Fonctions embarquent chacun une matrice, qui est une représentation discrète de la fonction mathématique qu'ils appliquent. La première ligne et la première colonne de la matrice sont les axes de la fonction, correspondant à chacun des arguments, tandis que les autres éléments sont les valeurs que prend la fonction. Le tableau 5.4 montre un exemple d'une telle matrice, représentant une fonction f à deux arguments x et y. On voit sur les axes (en gris) que $x \in [1;10]$ et $y \in [0;70]$. La valeur $f(x,y)$ varie quant à elle entre 0 et 100. Un Agent Fonction calcule la valeur de son port de sortie simplement en lisant l'élément de la matrice correspondant à la valeur de ses ports d'entrée (par exemple, toujours avec la fonction du tableau 5.4, $f(4,60) = 0$). Bien sur, il arrive souvent que la valeur des ports d'entrée ne corresponde pas directement à une case des axes. Dans ces cas-là, une interpolation linéaire est effectuée pour calculer la valeur de la fonction. Par exemple, $f(3.5,60) = 4.5$, ou encore $f(3.5,55) = 9$. L'auto-ajustement consiste donc à trouver la valeur adéquate pour chacun des éléments (axes compris) des matrices.

Les Agents Sorties et les Agents Entrées connaissent leur propre plage de variation, qui est directement définie par l'utilisateur. Chaque Agent Sortie envoie un message contenant sa

TABLE 5.4 – Un exemple de matrice embarquée par un Agent Fonction.

–	1	2	3	4	5	6	7	8	9	10
0	37	46	55	64	73	82	73	64	55	46
10	46	55	64	73	82	91	82	73	64	55
20	55	64	73	82	91	100	91	82	73	64
30	50	50	50	50	82	91	82	73	64	55
40	45	36	27	18	50	82	73	64	55	46
50	36	27	18	9	18	50	50	50	50	50
60	27	18	9	0	9	18	27	36	45	54
70	36	27	18	9	18	27	36	45	54	63

plage à l'Agent Fonction auquel il est lié. Ce dernier peut alors remplir la partie de sa matrice correspondant aux valeurs de $f(x, y)$. Il place aléatoirement dans la matrice les deux bornes de la plage de variation, et calcule la valeur v de chacun des autres éléments avec la formule suivante :

$$v = a * d + m$$

où d est la distance de Manhattan entre la case de l'élément calculé et celle de la borne m la plus proche. Le coefficient a est positif si m est la borne inférieure, négatif sinon. Si la valeur de v ainsi calculée sort de la plage de variation imposée, la valeur de la borne la plus proche lui est attribuée. Notons que tous les Agents Fonctions sont préalablement initialisés selon cette procédure, avec la même plage de variation, et des axes correspondant à cette plage. Ainsi, les branchements internes entre Agents Fonctions sont cohérents.

Une fois toutes les valeurs des éléments calculées, la plage de variation de $f(x, y)$ est la même que celle de l'Agent Sortie, et la sortie correspondante de la boîte noire est donc conforme aux attentes de l'utilisateur. L'Agent Fonction envoie alors, à tous les autres partenaires liés à son port de sortie (éventuellement lui-même), cette plage de variation. Parallèlement, chaque Agent Entrée envoie également sa propre plage de variation à tous les Agents Fonctions auxquels il est lié.

Enfin, tous les Agents Fonctions ayant reçu une plage de variation depuis un partenaire sur un port d'entrée l'utilisent pour remplir l'axe correspondant de leur matrice, en échantillonnant l'intervalle de manière uniforme. À l'issue de ce processus, les plages de variations des entrées et des sorties de la boîte noire respectent les spécifications de l'utilisateur.

La section suivante illustre la génération complète d'une boîte noire en déroulant un exemple simple.

5.3.4 Déroulement d'une génération simple

Dans cet exemple, l'utilisateur veut générer une boîte noire abstraite possédant deux entrées (qu'il nomme In1 et In2), deux sorties (Out1 et Out2), deux fonctions et aucun cycle.

Il désire que les entrées varient de 1 à 10 et les sorties de 0 à 100. Enfin, il précise que la sortie Out1 doit être influencée par les deux entrées, tandis que Out2 ne doit être influencée que par In2. La figure 5.3 montre une capture d'écran de l'interface permettant à l'utilisateur de spécifier ses exigences.

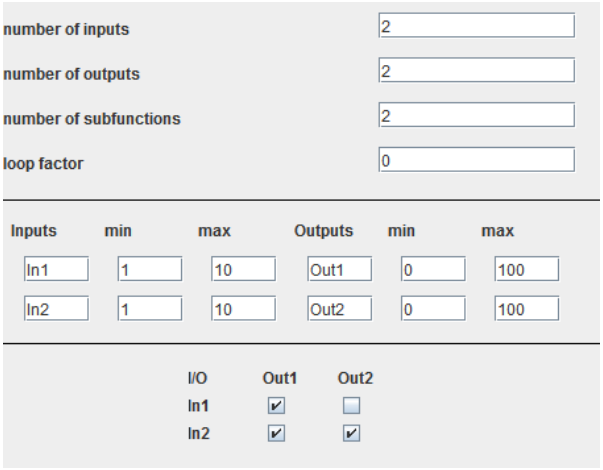

Figure 5.3 – Capture d'écran de l'interface de spécification d'une boîte noire.

Une fois que ces paramètres sont fixés, la génération peut commencer. Six agents sont créés :

- deux Agents Entrées, In1 et In2, initialisés avec une contrainte de lien chacun ;
- deux Agents Sorties, Out1 et Out2, chacun initialisé avec une contrainte de lien et une contrainte de chemin ;
- deux Agents Fonctions, F1 et F2, chacun initialisé avec une contrainte de lien, mais pas de contrainte de cycle (puisque le facteur de bouclage a été mis à 0).

Les contraintes de chemin de Out1 et Out2 sont directement déduites des interdépendances spécifiées par l'utilisateur. Ainsi celle de Out1 contient la liste $[In1; In2]$, et celle de Out2 la liste $[In2]$.

Dans BACH, c'est l'agent le plus critique qui a la priorité. Il s'agit ici de Out1, avec un niveau de criticité de 6 (une contrainte de lien avec un niveau de 2, plus une contrainte de chemin avec un niveau de 4). Le comportement associé à la contrainte la plus critique est déclenché (voir tableau 5.2). Aucun Agent Fonction n'est déjà lié à des Agents Entrées, c'est donc arbitrairement que F1 est choisi par Out1, qui se lie à son port de sortie. Ce lien provoque la résolution de la contrainte de lien de Out1 (et ce dernier délègue sa contrainte de chemin à F1) mais résout également la contrainte de lien en sortie de F1. Le niveau de criticité de Out1 est maintenant de zéro.

Le nouvel agent le plus critique est F1, qui vient de recevoir une contrainte de chemin. La liste de cette contrainte mentionne deux Agents Entrées à relier, et les deux ports d'entrées de F1 sont libres. F1 peut donc lier son premier port d'entrée à In1 et son second à In2. Ce faisant, plusieurs contraintes sont résolues : la contrainte de chemin reçue par F1, mais aussi

les contraintes de lien de F1, In1 et In2. La figure 5.4 montre l'état de la boîte noire à ce stade de l'auto-composition.

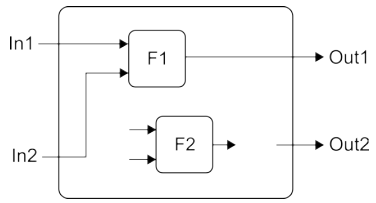

FIGURE 5.4 – Exemple de boîte noire abstraite en cours de génération.

L'agent le plus critique est maintenant Out2. Sa contrainte de chemin lui impose de relier In2. Out2 cherche donc à se lier à un Agent Fonction. F1 est déjà liée à In2, mais également à In1. Ce n'est donc pas un partenaire potentiel. Le choix de Out2 se porte donc sur F2. Out2 délègue sa contrainte de chemin à F2 après s'y être lié, et F2 devient l'agent le plus critique. Comme F2 a plus de ports d'entrées libres que d'Agents Entrées à rejoindre, il peut lier un de ses ports d'entrée directement à In2 et résoudre ainsi la contrainte de chemin. F2 est toujours l'agent le plus critique, car il est le seul à qui il reste une contrainte non résolue. Il s'agit de la contrainte de lien de ses ports d'entrée. F2 a le choix entre quatre possibilités pour lier son dernier port d'entrée : la sortie de F1, sa propre sortie, l'Agent Entrée In1 ou l'Agent Entrée In2. F2 n'a pas de contrainte de cycle, il ne doit donc pas lier une de ses entrées à sa propre sortie. En outre, les agents n'oublient pas les contraintes de chemin qu'ils ont déjà résolues. Aussi, le seul choix permettant à F2 de ne pas enfreindre les interdépendances souhaitées est de lier son deuxième port d'entrée à In2.

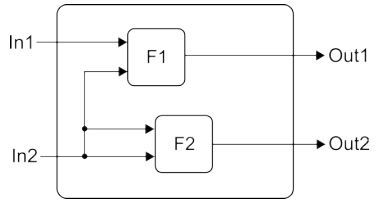

FIGURE 5.5 – Exemple de boîte noire abstraite après l'auto-composition.

Le processus d'auto-composition est maintenant terminé. La structure de boîte noire produite, illustrée par la figure 5.5, remplit toutes les spécifications de l'utilisateur, à l'exception des plages de variation.

Il n'y a pas de priorité entre les agents pour la phase d'auto-ajustement. Out1 envoie sa plage de variation $[0;100]$ à F1, qui place donc aléatoirement un élément de sa matrice à 0 et un autre à 100, puis remplit le reste des éléments selon la méthode décrite dans la section précédente. Parallèlement Out2 fait de même avec F2. Dans ce cas particulier, où les deux ports d'entrée sont liés au même partenaire, la matrice ne contient qu'un seul axe. In1 et In2 envoient également à leur(s) partenaire(s) leur plage de variation. F1 et F2 peuvent alors ajuster leurs axes, afin qu'ils couvrent les valeurs de 1 à 10. La boîte noire abstraite

est maintenant terminée et prête à l'emploi, les agents peuvent réaliser leur comportement nominal. BACH permet en outre de sauvegarder les boîtes noires sous forme de fichier XML et de générer des agents déjà liés et ajustés à partir de ces fichiers sauvegardés.

La boîte noire de cet exemple est bien trop simple pour être vraiment intéressante du point de vue du contrôle. La section suivante montre un exemple de boîte noire un peu plus conséquente.

5.4 Comportement des boîtes noires générées

Cette section montre un exemple de boîte noire abstraite générée par BACH, et présentant un éventail des différents comportements que l'on peut observer sur les sorties. Les effets des différents paramètres de la génération sont discutés.

L'utilisateur a donné les exigences suivantes pour la génération de la boîte de notre exemple :

- 4 entrées, 4 sorties et 11 fonctions ;
- la première entrée (E1) n'influe que sur la première sortie (S1), la seconde entrée (E2) sur les deux premières sorties (S1, et S2), la troisième entrée (E3) sur les trois premières sorties (S1 à S3), et enfin la quatrième entrée (E4) sur toutes les sorties (S1 à S4) ;
- 50% des fonctions sont dans un cycle ;
- toutes les variables varient entre 0 et 100.

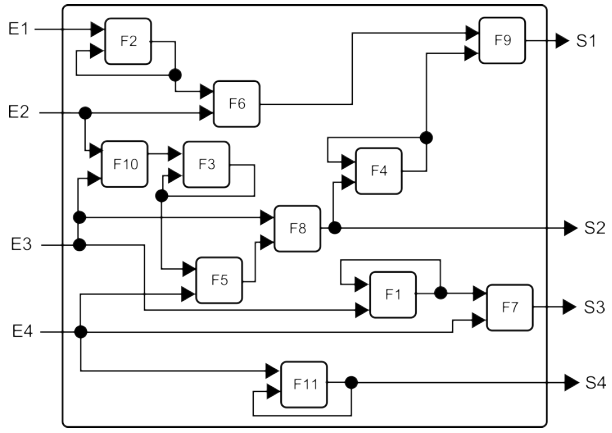

Figure 5.6 – Exemple de boîte noire abstraite générée.

La figure 5.6 montre le résultat de l'auto-composition. On peut voir que les interdépendances sont respectées, et que 5 fonctions bouclent sur elles-mêmes (F1, F2, F3, F4, et F11, soit 50% du nombre total de fonctions, arrondi à l'entier inférieur).

Lors de l'exécution d'une boîte, à chaque pas de temps, chaque fonction calcule sa nouvelle valeur et la transmet sur sa sortie. Aussi, une modification sur une entrée n'est pas instantanément répercutée sur les sorties concernées, la durée dépend du nombre de fonctions qui les relient. Plus le nombre de fonctions est grand, plus il faut de temps avant d'observer en sortie les conséquences d'une modification en entrée.

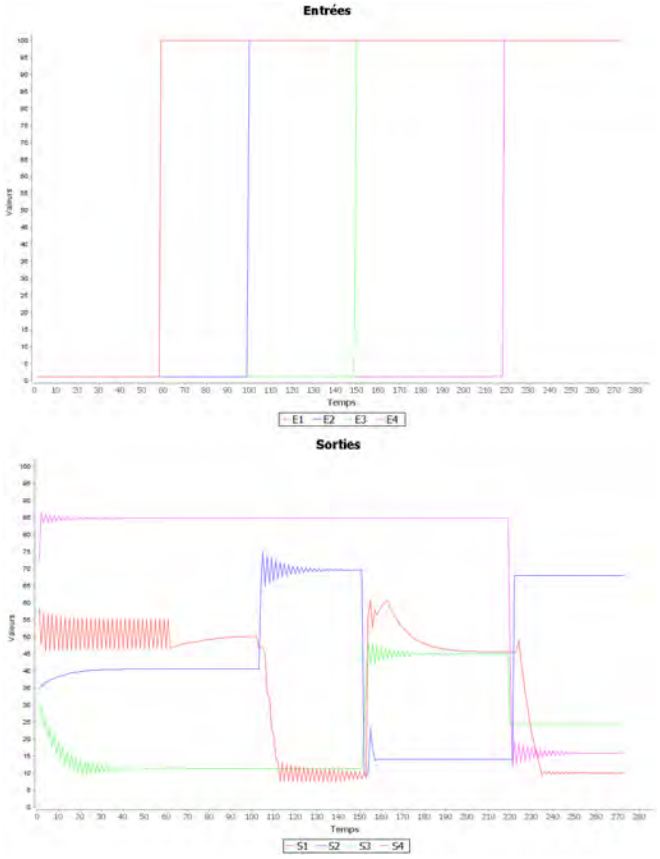

FIGURE 5.7 – Évolution des entrées et des sorties d'une boîte noire générée.

La figure 5.7 montre l'évolution des entrées et des sorties de la boîte de notre exemple. Les entrées sont toutes initialisées à 1. Lors des premiers pas de temps (dans notre cas, il s'agit des 58 premiers), la boîte s'initialise. Autrement dit, les valeurs des entrées influent progressivement les chaînes de fonctions, jusqu'à atteindre les sorties. La présence de cycles provoque des oscillations et des amortissements. Lors de la phase d'initialisation, nous voyons plusieurs exemples de ces phénomènes :

- S1 (en rouge) oscille autour d'une valeur stable, avec une amplitude constante ;
- S2 (en bleu) est "amortie", elle se rapproche d'une valeur de plus en plus lentement ;
- S3 (en vert) oscille autour d'une valeur qui converge de plus en plus lentement, en outre, l'amplitude des oscillations diminue avec le temps ;
- S4 (en magenta) montre des oscillations amorties autour d'une valeur stable ;

Au 58^e pas de temps, E1 est passée à 100. Conformément aux spécifications, S1 est la seule sortie impactée. En effet, quatre pas de temps plus tard, ses oscillations stoppent et elle se met à converger lentement vers la valeur 50.

Au pas de temps 99, E2 est passée à 100, ce qui provoque des changements sur S1 et S2 (respectivement 3 et 4 pas plus tard). La valeur de S1 diminue fortement pendant quelques pas, avant d'osciller périodiquement autour d'une valeur stable. S2 augmente se met à osciller

de plus en plus faiblement.

Une fois les oscillations de S2 devenues imperceptibles, c'est au tour de E3 d'être mise à la valeur 100. S1 augmente fortement, puis oscille de manière irrégulière, avant de converger vers la valeur 46. S2 diminue, oscille une fois avant d'être complètement stable. Enfin, S3 augmente, puis oscille de manière amortie autour de 45.

Enfin, c'est la valeur de E4 qui est modifiée. Cette fois-ci, toutes les sorties sont impactées. S1 augmente un petit peu avant de plonger vers la valeur 10, autour de laquelle elle oscille de plus en plus faiblement. S2 augmente et S3 diminue, aucune ne présente d'oscillation. S4 diminue dès le pas suivant, puis montre des oscillations dont l'amplitude diminue.

Cet exemple illustre les caractéristiques essentielles que peuvent présenter les boîtes noires générées par BACH : une entrée influant sur plusieurs sorties, une sortie dépendante de plusieurs entrées, des comportements non linéaires ainsi qu'une certaine latence.

5.5 Résumé

Ce chapitre a présenté BACH, un système multi-agent de génération de boîtes noires abstraites. Ces boîtes noires sont utilisées en remplacement d'un simulateur afin de tester les performances d'un contrôleur capable d'apprentissage pendant son développement, avant de le déployer sur le système ciblé. Elles permettent de se concentrer sur différentes manifestations de la complexité (ou sur des combinaisons de celles-ci), et de disposer d'un large corpus de cas de test.

Il se pose néanmoins la question de savoir si ces boîtes noires sont d'une complexité suffisante pour que les tests que l'on effectue avec soient pertinents. Un moyen de vérifier cela est de développer un contrôleur en procédant à sa mise au point à l'aide de boîtes générées par BACH, puis de le tester sur le véritable procédé auquel il est destiné. Si le contrôleur fonctionne aussi bien (et sans avoir été lourdement modifié) sur le procédé réel que sur les boîtes noires, alors ces dernières sont adéquates.

C'est précisément ce qui a été fait avec ESCHER. Aussi, les expérimentations présentées dans le chapitre suivant, qui consistent en l'application de notre système de contrôle multi-agent sur des boîtes générées, ainsi que sur un vrai moteur à combustion, constituent une validation de ESCHER, mais également de BACH.

CHAPITRE 6

Expérimentations

Ce chapitre présente les résultats obtenus avec ESCHER, d'abord sur des boîtes noires générées par BACH, puis sur un véritable moteur à combustion. Ces expérimentations visent à montrer comment ESCHER converge vers un contrôle adéquat, et parvient à placer dans un état désiré un système à propos duquel il n'a aucune connaissance préalable. Les résultats de ces expériences sont ensuite discutés.

6.1 Expériences sur boîtes noires générées

Les expérimentations présentées dans cette section ont été conduites sur diverses boîtes noires abstraites générées par BACH (voir chapitre 5). L'évolution des sorties, les modifications effectuées par ESCHER sur les entrées, ainsi que les variations sur les niveaux de criticité sont illustrées sur plusieurs cas de complexité croissante. Pour ces expériences, un cycle de vie correspond à l'exécution d'un cycle de vie de chacun des agents de ESCHER, suivi d'un pas d'exécution effectué par la boîte noire contrôlée. La durée d'un cycle de vie de ESCHER est variable selon le nombre d'agents, mais demeure de l'ordre de la milliseconde.

Dans ESCHER, chaque critère de contrôle (consigne, seuil, ou optimisation) est associé à un niveau de criticité qui reflète sa satisfaction. Lorsque le critère est pleinement satisfait, sa criticité est nulle. Aussi, cette dernière donne une évaluation du résultat obtenu par le contrôleur. La fonction utilisée est décrite en appendice.

6.1.1 Atteindre une consigne sur une boîte SISO

Pour cette première expérience, ESCHER est confronté à une boîte noire comprenant une seule entrée (variant de 1 à 10) et une seule sortie (variant de 0 à 100). Le système reçoit la consigne de placer la sortie à 50. Il démarre vierge d'Agent Contexte, il n'a donc aucune information sur le comportement de la boîte noire.

La figure 6.1 montre l'évolution de l'entrée et de la sortie de la boîte noire, ainsi que la variation du nombre de d'Agents Contextes, et celle du niveau de criticité. L'entrée de la boîte

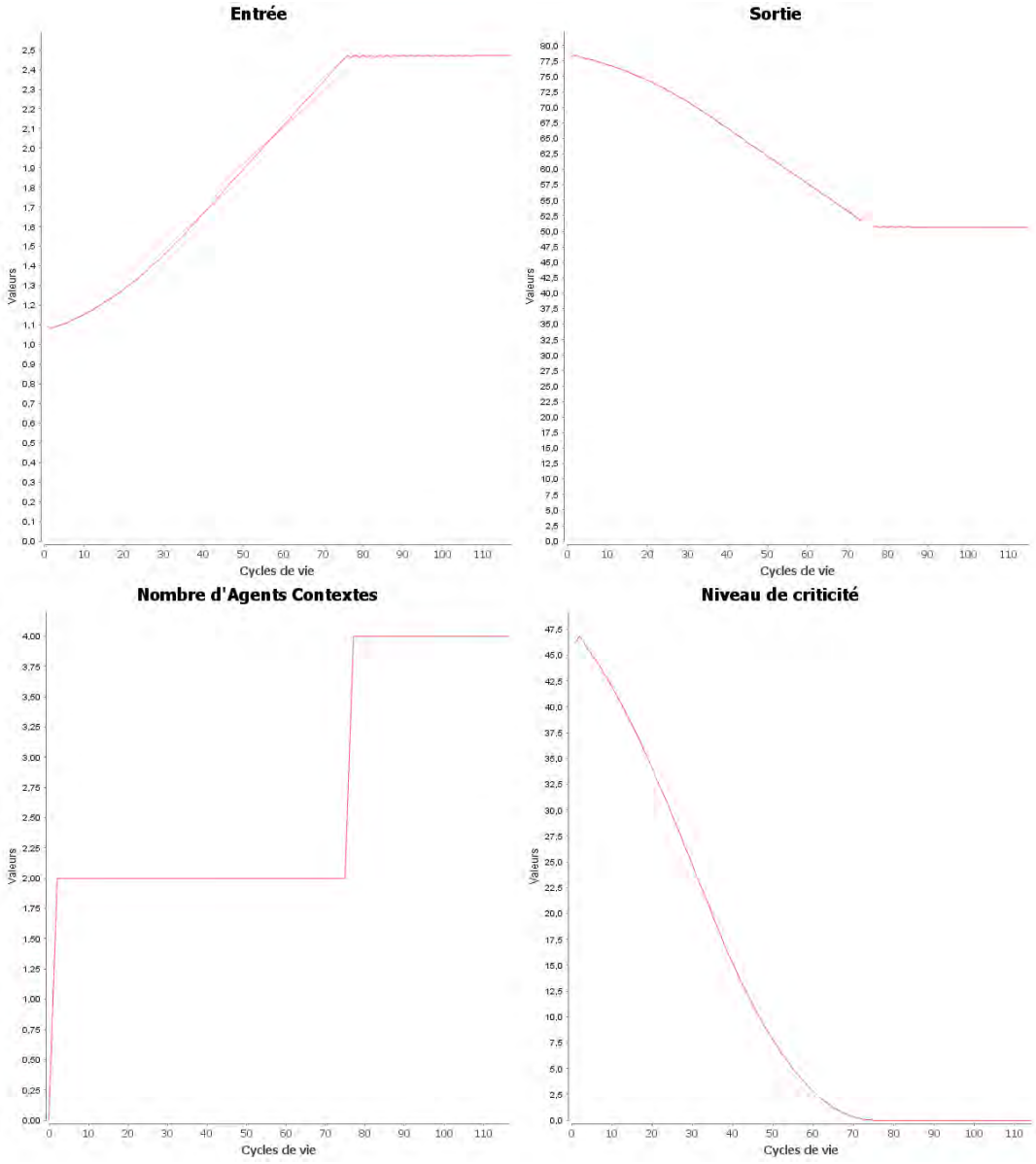

Figure 6.1 – Contrôle d'une boîte SISO par ESCHER.

est initialisée à 1.1, ce qui place sa sortie à la valeur 78.2, et entraîne un niveau de criticité de 46.22. Au départ, comme il n'a aucune connaissance, ESCHER effectue une action choisie aléatoirement : il fait légèrement diminuer l'entrée. Un Agent Contexte correspondant à cette action est créé. Cependant, diminuer l'entrée était une erreur, car cela a éloigné la sortie de la consigne, et donc provoqué une augmentation de la criticité. C'est pourquoi l'action n'est pas conservée, et ESCHER commence alors à augmenter l'entrée. Un autre Agent Contexte est créé. Cette fois-ci, le niveau de criticité diminue. Aussi, l'Agent Contrôleur en charge de l'entrée poursuit cette même action et en informe le nouvel Agent Contexte. Ce dernier se

retrouve en SNC 9 et va donc ajuster l'action qu'il propose. Il en augmente l'amplitude. On observe alors que l'entrée de la boîte noire augmente de plus en plus rapidement, tandis que la sortie se rapproche de sa consigne en accélérant, et que la criticité diminue. Puisque la même action permet d'atteindre la consigne, aucun Agent Contexte n'est créé jusqu'à ce que le niveau de criticité soit de zéro (cela survient au cycle de vie 74).

À ce moment, l'action effectuée par l'Agent Contrôleur (toujours sur proposition du deuxième Agent Contexte) ne provoque pas de diminution de la criticité, contrairement aux prévisions. Les SNC 3 et 4 sont alors déclenchées. L'Agent Contrôleur décide d'appliquer l'action opposée, c'est-à-dire de baisser la valeur de l'entrée. Un nouvel Agent Contexte est créé. Cependant, près de son minimum, la fonction de criticité est assez plate. Aussi, cette nouvelle action ne diminue pas la criticité. L'Agent Contrôleur décide donc de changer d'action (il augmente maintenant l'entrée), et crée un quatrième Agent Contexte.

Ces deux derniers Agents Contextes sont alors alternativement sélectionnés, stabilisant la sortie autour de la consigne. L'ajustement de l'amplitude de l'action tend à réduire ces oscillations. ESCHER se stabilise ainsi dans une configuration à quatre Agents Contextes. L'entrée de la boîte noire vaut 2.48 tandis que la sortie vaut 50 (conformément à la consigne).

Ce cas de petite dimension, très simple, nous permet d'illustrer graphiquement la manière dont les Agents Contextes se répartissent sur l'espace d'états du procédé, et comment l'ensemble des prévisions peut donner une approximation linéaire par partie, et partielle, de la fonction de criticité.

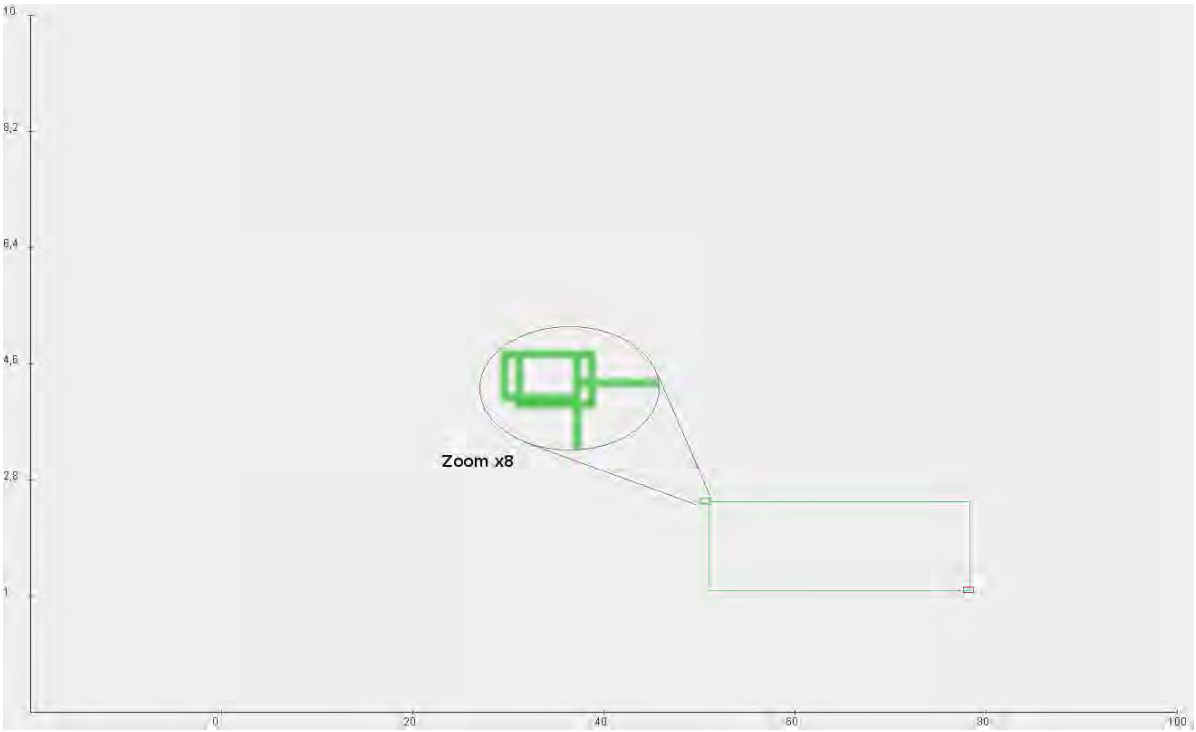

FIGURE 6.2 – Illustration des plages de validité des Agents Contextes.

La figure 6.2 est une capture d'écran montrant les plages de validité des Agents Contextes dans l'espace en deux dimension formé par les deux variables de la boîte noire (l'entrée et la sortie). En bas à droite, on trouve le premier Agent Contexte créé, dont l'action a fait augmenter la criticité (en rouge). Le gros rectangle est formé par les plages de validité du deuxième Agent Contexte, qui a été conservé durant toute la convergence vers la consigne, et a donc eu l'occasion de s'étendre. Enfin, les deux petits rectangles partiellement superposés sont les deux derniers Agents Contextes créés. On voit que ESCHER a uniquement parcouru la zone de l'espace d'états du procédé qui mène vers la consigne, sans chercher à explorer le reste.

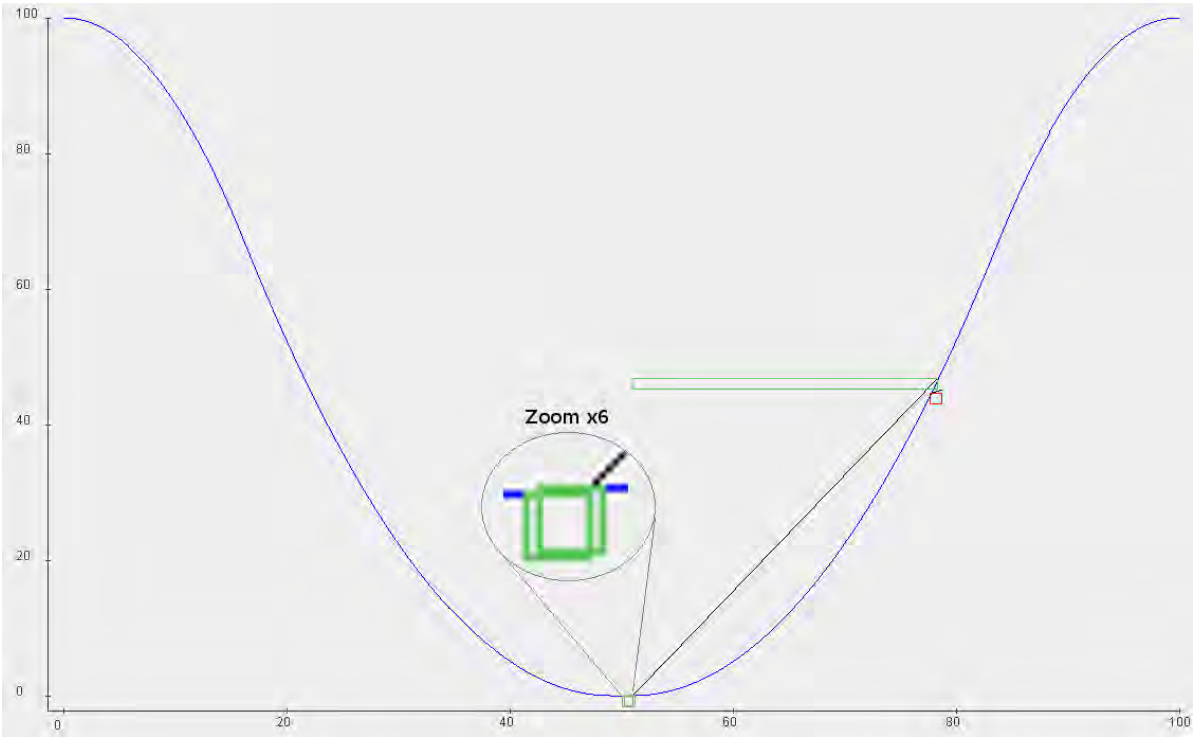

Figure 6.3 – Illustration des prévisions des Agents Contextes.

La figure 6.3 montre une capture d'écran avec la courbe de la fonction de criticité. Son minimum vaut 0, il est atteint lorsque la sortie de la boîte vérifie la consigne (50 dans notre cas). Sur cette courbe sont projetées les plages de validité (uniquement celles concernant la sortie), et les prévisions des Agents Contextes. Les plages sont représentées par la longueur des rectangles et leur positionnement par rapport à l'axe des abscisses, tandis que les prévisions sont représentées par les droites noires. On remarque à droite le premier Agent Contexte, rouge, avec sa prévision indiquant une augmentation de criticité. Le grand rectangle est notre deuxième Agent Contexte, on voit qu'il a correctement appris la diminution de criticité que son action provoque, la droite atteignant le minimum de la courbe. Enfin, les deux rectangles partiellement superposés, autour du minimum, sont les deux agents qui provoquent la stabilisation autour de la consigne.

6.1.2 Atteindre un compromis

La difficulté de ce cas est légèrement supérieure au précédent. La boîte à contrôler comprend une entrée qui influe cette fois-ci sur deux sorties (S1 et S2). La consigne est de placer les deux sorties, qui varient de 0 à 100, à la valeur 50. Il y a donc deux Agents Critères, un pour chaque sortie. Chacun embarque la même fonction de criticité. Ainsi, les deux sorties ont la même importance. Cependant, cette consigne n'est en fait pas réalisable, il n'existe pas de valeur en entrée qui place les deux sorties à 50. ESCHER doit découvrir le bon compromis, c'est-à-dire la valeur de l'entrée pour laquelle le plus haut des deux niveaux de criticité est minimal.

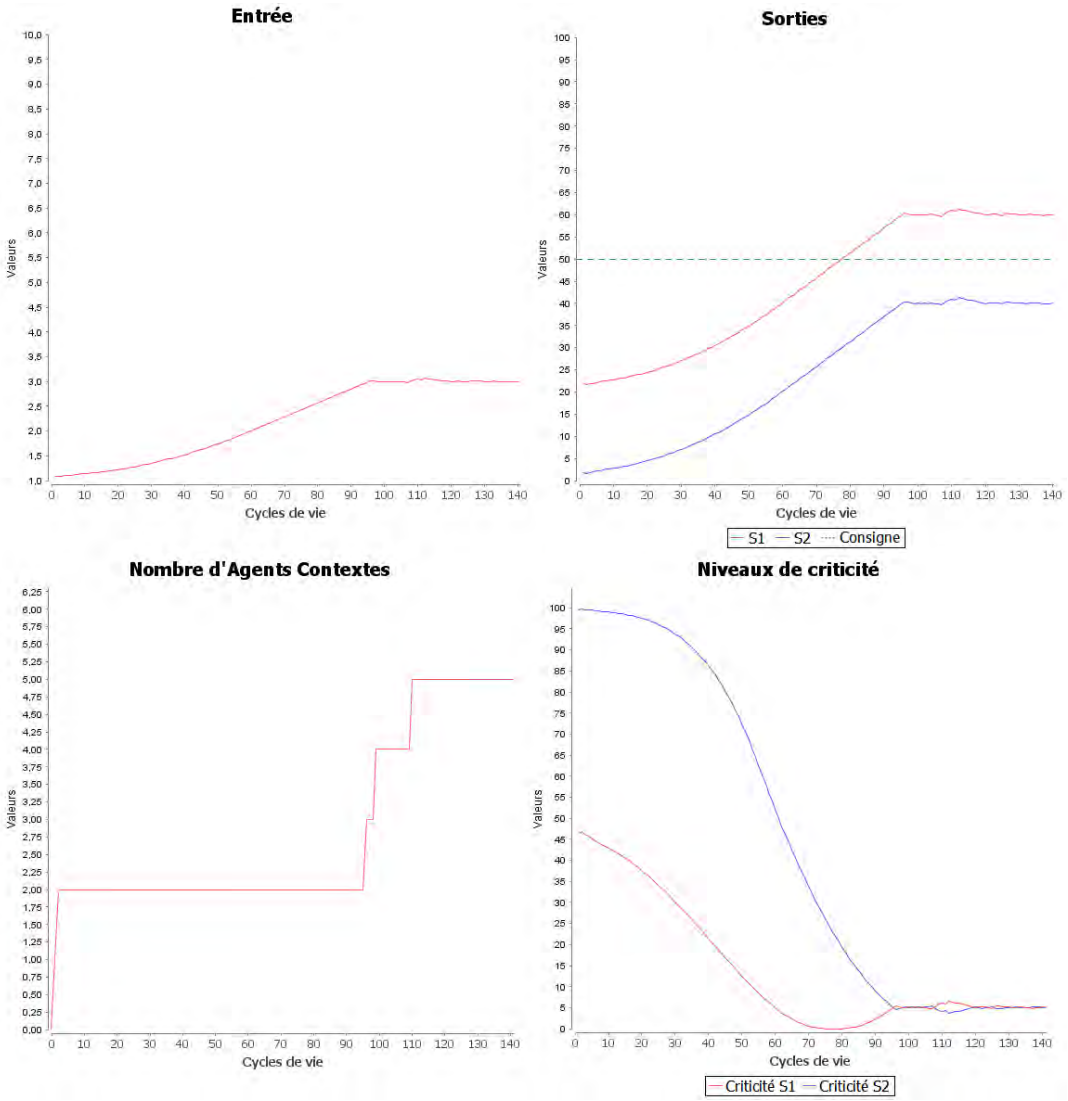

FIGURE 6.4 – Obtention d'un compromis entre deux consignes.

La figure 6.4 montre l'évolution de l'entrée et des sorties de la boîte contrôlée, du nombre

d'Agents Contextes, et des niveaux de criticité. L'entrée est initialisée à 1.1, ce qui place S1 à 21.8 et S2 à 1.8. S2 étant plus éloignée de la consigne, le niveau de criticité qui lui est associé est plus élevé que celui correspondant à S1. Comme lors de l'expérience précédente, ESCHER n'a aucune connaissance préalable du fonctionnement de la boîte qu'il contrôle. Sa première action est une erreur, il diminue la valeur de l'entrée et provoque ainsi une légère hausse des niveaux de criticité. ESCHER corrige cette erreur dès le pas suivant, et commence à augmenter la valeur de l'entrée.

Ainsi, deux Agents Contextes ont été créés dès les premiers cycles de vie de l'Agent Contrôleur. La deuxième action appliquée permettant de diminuer la criticité, elle est conservée, et l'Agent Contexte qui la propose suffit à ESCHER. Ainsi, on observe que le nombre d'Agents Contextes dans le système reste stable tant que le niveau de criticité maximum diminue (c'est-à-dire entre les cycles de vie 2 et 95).

Le niveau de criticité de la consigne sur S1 atteint 0 au cycle 76. Cependant, le niveau de criticité de la consigne de S2 est alors à 26.1, et il est toujours possible de le faire diminuer. Aussi, ESCHER poursuit son action, malgré l'augmentation du niveau de criticité de la consigne de S1. L'action est maintenue tant que ce dernier reste inférieur à celui de la consigne de S2, autrement dit, tant que le maximum des niveaux de criticité diminue. La sortie S1 dépasse alors la consigne de 50 et commence à s'en éloigner, tandis que S2 continue de s'en rapprocher.

C'est au cycle de vie 96 que la criticité de S1 devient plus importante que celle de S2. ESCHER change alors d'action, et les niveaux de criticité se croisent à nouveau. Il s'ensuit une série d'oscillations, au cours desquelles trois nouveaux Agents Contextes sont créés. Finalement, la valeur de l'entrée se stabilise, oscillant de manière très légère autour de 3, tandis que S1 fait de même autour de 60, et S2 autour de 40. Les deux niveaux de criticité oscillent autour de 5. ESCHER a atteint le meilleur compromis, le niveau de criticité maximal est le plus bas possible, même si aucune des sorties n'a atteint sa consigne.

Cette expérience montre comment un Agent Contrôleur est capable de gérer une entrée qui impacte plusieurs sorties, avec des consignes contradictoires. Notons que des fonctions de criticité différentes peuvent mener à un compromis différent. On peut par exemple donner la priorité à une sortie, quitte à dégrader l'autre, en faisant en sorte que son niveau de criticité soit plus important. Ce point est discuté dans la section 6.3.

6.1.3 Contrôler plusieurs entrées

La boîte noire utilisée dans cette expérience possède deux entrées (E1 et E2) pouvant varier entre 0 et 10, et une sortie qui varie entre 0 et 100. La consigne est de placer cette sortie à 50. Ici, ESCHER comprend donc deux Agents Contrôleurs qui vont devoir agir ensemble, en parallèle et de manière cohérente, pour que la sortie atteigne la consigne. Comme d'habitude, ESCHER démarre sans aucune connaissance du comportement de la boîte noire. Les Agents Contrôleurs doivent donc faire sans Agent Contexte pour commencer.

La figure 6.5 montre l'évolution des entrées et de la sortie de la boîte, du nombre total d'Agents Contextes dans le système, du nombre d'Agents Contextes valides simultanément,

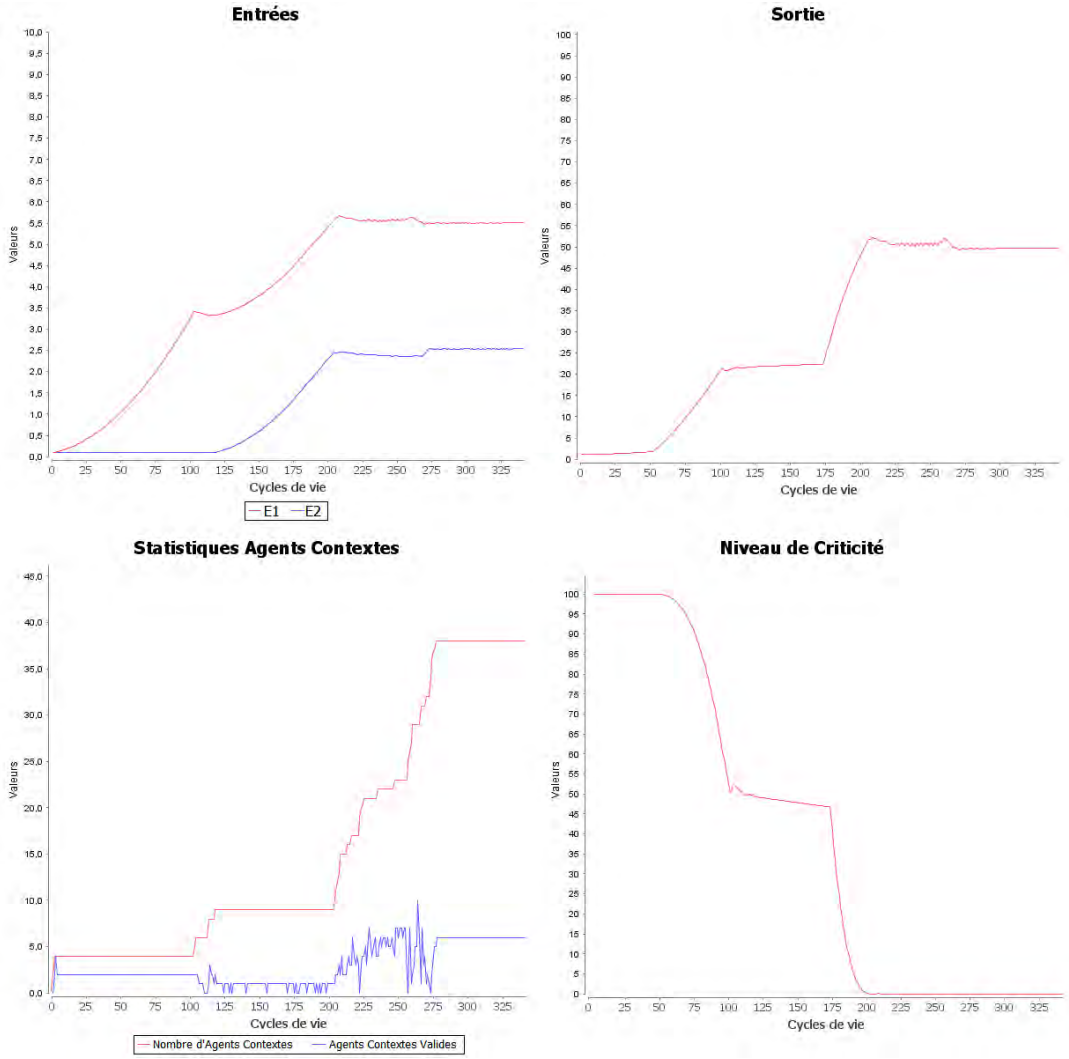

FIGURE 6.5 – Contrôle de deux entrées pour atteindre une consigne sur une sortie unique.

ainsi que la variation de niveau de criticité.

C'est l'Agent Contrôleur de E1 qui tente une action en premier, tandis que celui de E2 choisit de ne pas modifier son entrée. Cela provoque une légère variation de la sortie, qui se rapproche de la consigne. Puisque la criticité diminue, chacun des Agents Contrôleurs maintient son action courante. D'abord faible, la variation de la sortie (comme celle du niveau de criticité) accélère brusquement au cycle de vie 52. Cela ne perturbe pas ESCHER qui maintient les actions. Au centième cycle de vie, la sortie change de sens de variation, et la criticité remonte. L'Agent Contrôleur de E1 inverse donc son action, et la criticité reprend sa baisse. L'Agent Contrôleur de E2 n'a pas eu à modifier son action.

Mais la baisse ne se poursuit qu'une quinzaine de cycles, après quoi les Agents Contrôleurs sont forcés de changer d'action. E1 oscille brièvement, tandis que E2 est augmentée. La baisse de criticité reprend, faible dans un premier temps, avant d'accélérer à nouveau brutalement.

La sortie atteint la consigne, et la dépasse même légèrement. Les entrées oscillent alors, tout en étant ajustées (on observe par exemple un petit saut de E2), et le système de stabilise avec une criticité nulle et la consigne atteinte.

Cette expérience montre comment deux entrées peuvent être coordonnées par ESCHER pour atteindre une consigne sur une sortie. On note que la création d'Agents Contextes correspond aux moments où la variation de criticité change de sens. En effet, cela force les Agents Contrôleurs à trouver une nouvelle action, et donc à créer un Agent Contexte. La phase de stabilisation, pendant laquelle le système oscille, est celle qui voit le plus de nouveaux Agents Contextes apparaître. À la fin de l'expérience, il y en a un total de 38 dans le système (21 pour l'Agent Contrôleur de E1, 17 pour celui de E2). Cependant, le nombre d'Agents Contextes valides simultanément reste faible, avec un pic maximal à 10 lors de la phase de stabilisation (5 pour chaque Agent Contrôleur).

6.1.4 Contrôler une boîte MIMO

Cette expérience rassemble les difficultés des précédentes. La boîte noire utilisée possède quatre entrées (E1 à E4) variant de 0 à 10, et quatre sorties (S1 à S4) variant de 0 à 100. La consigne est de placer chaque sortie à la valeur 50. Il y a donc quatre Agents Critères dans le système, embarquant chacun une fonction de criticité identique. Ainsi, ESCHER doit contrôler plusieurs entrées pour faire respecter une consigne à plusieurs sorties. Or, du fait de la boîte noire choisie, il est impossible que toutes les sorties respectent la consigne. ESCHER doit néanmoins trouver la valeur à donner à chaque entrée pour que le niveau de criticité maximum soit le plus bas possible, c'est-à-dire atteindre un compromis dans lequel toutes les sorties sont le plus proche possible de la consigne.

La variation des entrées et des sorties de la boîte, ainsi que l'évolution du nombre d'Agents Contextes et celle des niveaux de criticité, sont illustrées par la figure 6.6. Les quatre entrées sont initialisées à 5. Cela place la sortie S2 à la consigne (sa criticité est donc nulle), tandis que S3 est la plus éloignée de la consigne (son niveau de criticité est donc le plus haut). Ainsi, ESCHER cherche d'abord à diminuer le niveau de criticité associé à la consigne sur S3, même si cela doit dégrader les autres.

Lors de la première dizaine de cycles de vie, ESCHER teste des actions de petite amplitude. Puis, une diminution plus importante de E1, accompagnée d'une augmentation de E4, provoque une baisse des niveaux de criticité de S3 (qui est actuellement le plus fort) et de S4. Au cycle 140, E1 et E4 ne varient plus que faiblement, et ce sont E2 (diminution) et E3 (augmentation) qui accélèrent. Cela a pour effet de poursuivre la baisse du niveau maximal de criticité.

Entre les cycles 200 et 300, la criticité se stabilise alors que les entrées oscillent autour de leur valeur courante. Ce n'est que lorsque E2 et E4 prennent simultanément la bonne direction (E2 diminue, et E4 augmente) que la criticité maximale entame une forte diminution (cycle 315). Cependant, cela s'accompagne d'une importante hausse du niveau de criticité de S4, qui finit par devenir le plus critique. Aussi, E2 cesse de diminuer et E4 d'augmenter. C'est la baisse de E1 qui permet finalement de diminuer à la fois les niveaux de criticité de S3 et

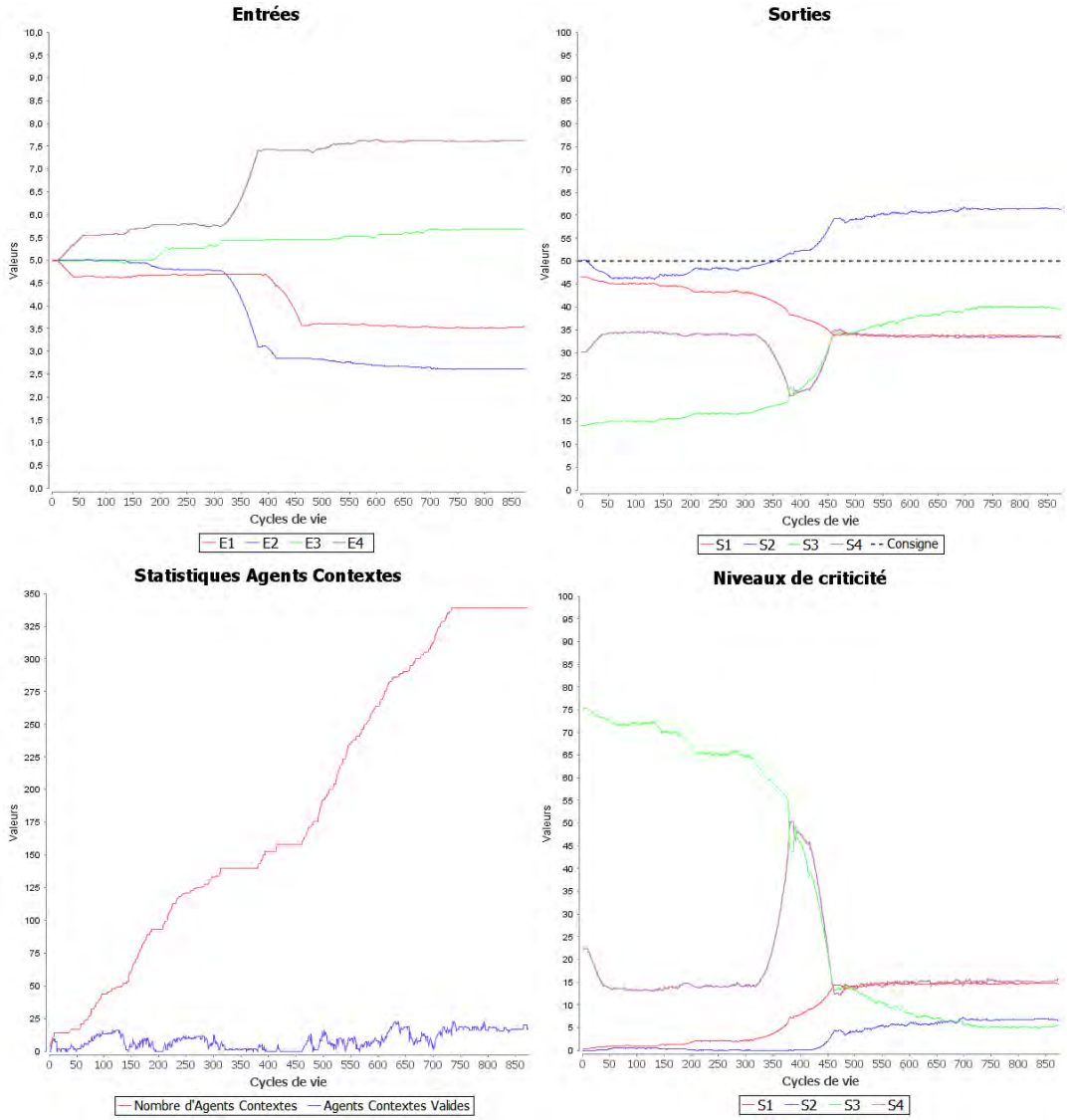

FIGURE 6.6 – Contrôle d'une boîte MIMO par ESCHER.

S4, moyennant une augmentation de ceux de S1 et S2. Au cycle 460, le niveau de criticité de S2 devient aussi important que ceux de S3 et S4. Les entrées se stabilisent alors, seules E2 et E3 continuent de légèrement varier. Cela permet à la criticité de S3 de poursuivre sa baisse, tandis que le niveau de criticité maximal reste stable.

À partir de 700 cycles environ, toutes les variables (entrées, sorties, et niveaux de criticités) sont stables, et le nombre d'Agents Contextes n'augmente plus. Il y a 339 Agents Contextes dans le système (88 pour l'Agent Contrôleur de E1, 83 pour E2, 76 pour E3, et 92 pour E4). On remarque que le nombre d'Agents Contextes valides simultanément reste faible, le maximum étant de 22 (en cumulé sur les quatre groupes présents dans ESCHER).

Toutes les sorties ont la même fonction de criticité. Elles ont donc la même importance

les unes par rapport aux autres. À la fin du test, S3 et S4 se sont rapprochées de la consigne, alors que S1 et S2 s'en sont éloignées. La forte amélioration de la situation de S3 a nécessité de dégrader légèrement celles de S1 et S2. L'enveloppe des quatre sorties est tout de même globalement plus proche de la consigne. Cela se traduit par un niveau de criticité maximal plus faible (de 75 à l'initialisation, celui-ci est tombé à 15). ESCHER est parvenu à trouver les valeurs des entrées permettant d'atteindre un compromis en sortie dicté par les fonctions de criticité.

6.1.5 Robustesse aux perturbations

Cette expérience montre comment ESCHER réagit aux perturbations de son environnement. Il contrôle ici deux des trois entrées d'une boîte noire (E1 et E2), la dernière (E3) étant

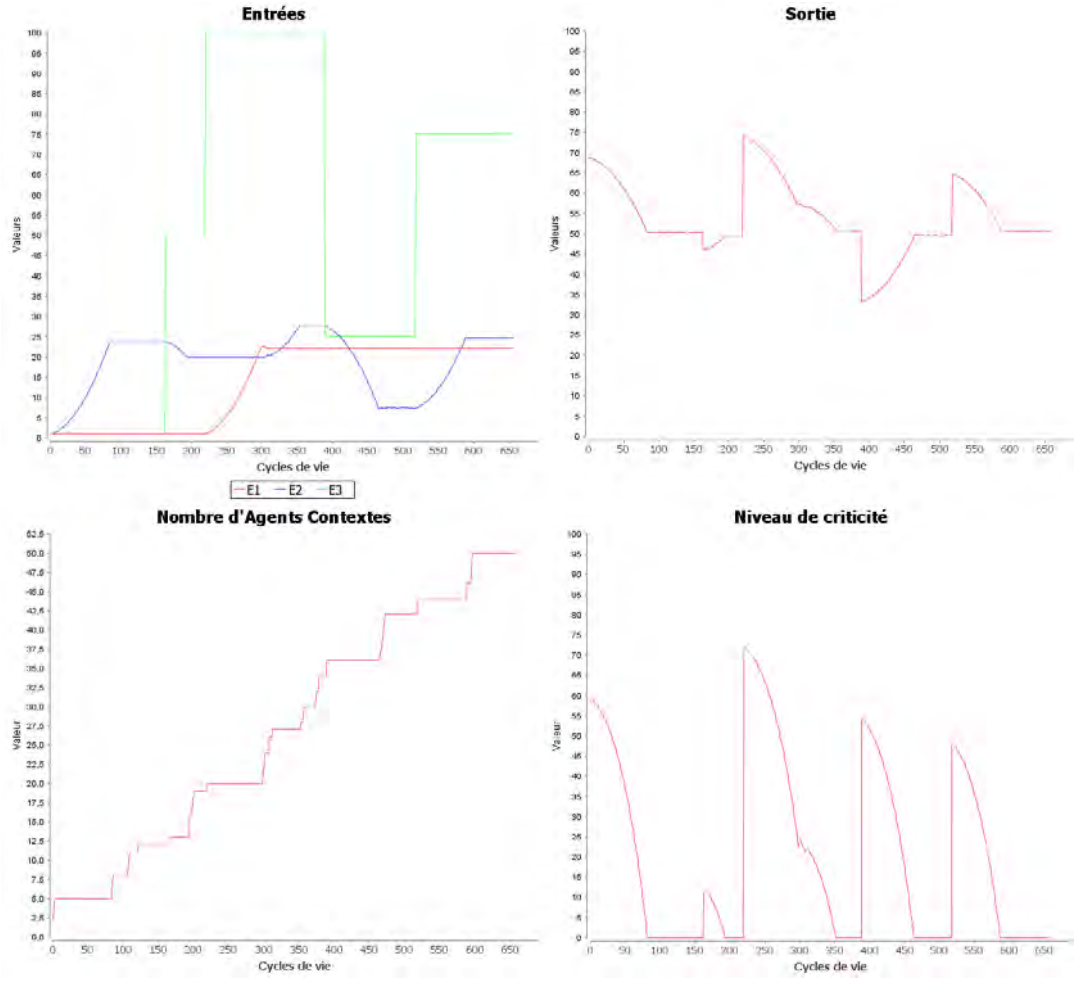

Figure 6.7 – Contrôle d'une boîte noire avec perturbations.

gérée manuellement. Ces trois entrées influencent la même sortie S1 de la boîte, à laquelle est associée une consigne. On laisse d'abord ESCHER faire converger S1 vers la consigne

en modifiant les valeurs de E1 et E2. On change ensuite la valeur de E3, provoquant une perturbation sur la sortie, qui s'éloigne brusquement de la consigne. ESCHER doit alors s'adapter à cette modification en trouvant de nouvelles valeurs pour les entrées qu'il contrôle.

La figure 6.7 montre l'évolution des entrées et de la sortie de la boîte (toutes varient entre 0 et 100), ainsi que la variation du nombre d'Agents Contextes et du niveau de criticité. Les entrées sont initialisées à 1, ce qui place la sortie à 68. La consigne est de placer cette dernière à 50. La consigne est atteinte en augmentant seulement E2, sans toucher à E1.

Au cycle 160, E3 est manuellement placée à 50, ce qui déloge S1 de sa consigne en provoquant une diminution. Cela se traduit par un pic du niveau de criticité, qui passe de 0 à 12. Cet écart est résorbé par ESCHER qui diminue E2 jusqu'à ce que la sortie respecte de nouveau la consigne.

E3 est à nouveau modifiée au cycle 220, elle passe de la valeur 50 à la valeur 100. Cela provoque une forte hausse de la sortie, et donc un deuxième pic de criticité (qui atteint 72). ESCHER augmente d'abord E1, puis E2. La criticité est ainsi ramenée à zéro, la sortie atteignant la consigne de 50 au cycle 350. Deux autres modifications de E3 (aux cycles 390 et 515) entraînent des augmentations du niveau de criticité, auxquelles ESCHER s'adapte en ajustant E2, et parvenant toujours à respecter la consigne.

Cette expérience montre que ESCHER est capable de réagir aux perturbations sur le système qu'il contrôle. Il s'adapte aux changements pour maintenir un contrôle adéquat. Ici, chaque perturbation est suffisamment importante pour provoquer la création de nouveaux Agents Contextes.

6.1.6 Répétabilité des expériences

L'objectif est ici de vérifier que l'on obtient des résultats équivalents sur un grand nombre de tests identiques. Aussi, nous montrons le résultat de 100 tests, tous effectués sur la même boîte noire. Lors de chaque test, ESCHER et la boîte noire démarrent à partir du même état initial, et 2000 cycles sont effectués. Afin d'alléger la présentation, nous nous concentrons uniquement sur l'évolution du niveau de criticité maximum, qui est le critère de mesure le plus pertinent pour évaluer la réussite d'un test.

Le test effectué concerne le contrôle d'une boîte à 10 entrées et 10 sorties. Toutes les sorties varient entre 0 et 100, la consigne étant de placer chacune à 50. Outre le nombre de variables, une difficulté pour ESCHER provient ici du fait que toutes les entrées n'influent pas sur toutes les sorties. Certains Agents Contrôleurs risquent donc d'observer des variations du niveau maximum de criticité qui ne sont pas de leur fait.

La figure 6.8 montre l'évolution de la criticité maximale pour chacun de ces 100 tests. La courbe rouge représente la moyenne, et la courbe bleue la médiane. On voit que la criticité maximale est toujours diminuée par ESCHER au cours du temps. On peut néanmoins voir que toutes les courbes sont différentes. En effet, en l'absence de proposition d'action (comme c'est le cas lors du démarrage du système), un Agent Contrôleur choisit aléatoirement la modification à appliquer à l'entrée qu'il contrôle. Aussi, chaque test prend un départ différent, et donc évolue différemment par la suite. Malgré tout, toutes les courbes convergent vers

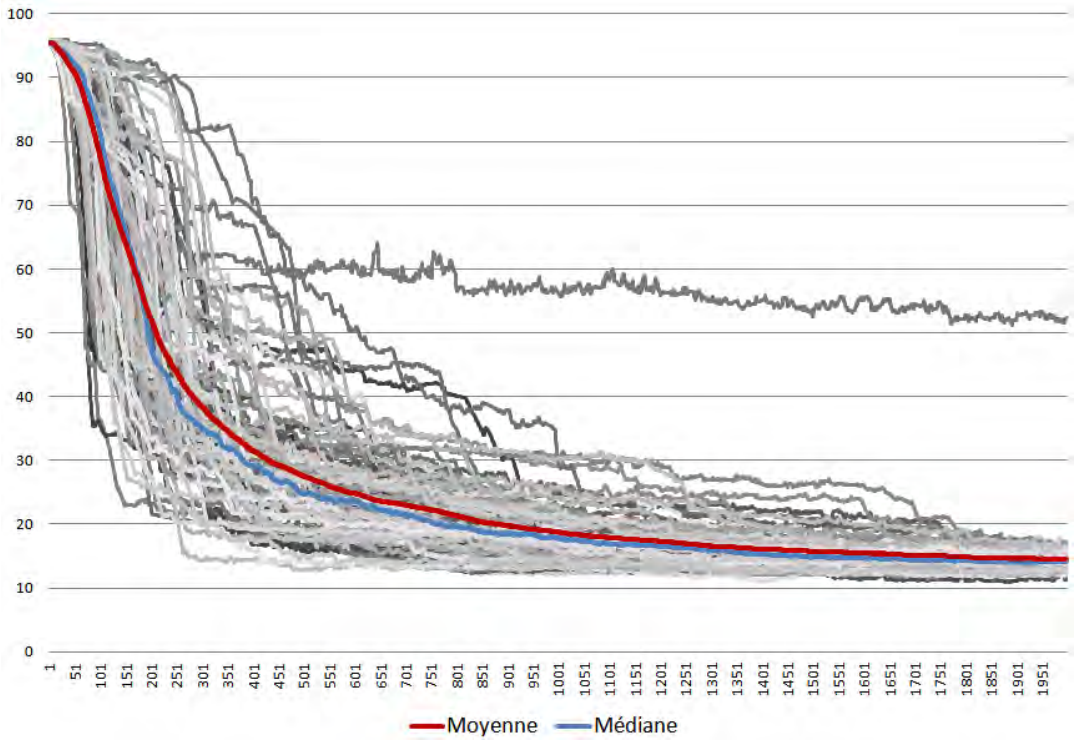

Figure 6.8 – Niveau de criticité maximale de 100 tests sur une boîte à 10 entrées et 10 sorties.

le même niveau de criticité (autour de 15), ce qui indique que ESCHER parvient bien à amener le procédé dans l'état souhaité, quelles que soient les erreurs qu'il commet lors de son apprentissage.

Un élément peut toutefois interpeller : un test semble avoir échoué, son niveau de criticité reste supérieur à 50 tout au long des 2000 cycles, ce qui est bien plus élevé que les 99 autres. On peut cependant voir que son niveau de criticité continue à diminuer. Il est probable qu'un allongement du nombre de cycles lui aurait permis de converger vers la même valeur que les autres. En effet, il est fréquent, avec les boîtes noires, que le niveau de criticité stagne pendant un certain nombre de cycles, avant que ESCHER ne trouve la solution pour le diminuer rapidement. Il se peut également que les erreurs précédentes commises par ESCHER sur ce test aient conduit la boîte noire dans un état d'où on ne peut sortir sans augmenter le niveau maximal de criticité (ce que ESCHER s'interdit de faire). Le système est ainsi dans un minimum local. Cela rejoint une des limites identifiées de ESCHER (voir section 6.3.2).

6.1.7 Bilan des expériences sur boîtes noires générées

Cette section a présenté un ensemble d'expériences réalisées sur des boîtes noires générées par BACH, représentatives des tests qui ont été menés lors du développement de ESCHER. Elles montrent que ce dernier est capable d'apprendre le contrôle de plusieurs entrées, pour atteindre des consignes sur plusieurs sorties (impliquant éventuellement des compromis).

La section suivante montre les résultats obtenus sur un banc d'essai moteur. Dans ces expériences, ESCHER doit trouver le moyen d'optimiser certaines sorties (comme maximiser le couple moteur), tout en respectant des critères parfois antinomiques sur d'autres (comme limiter les émissions de gaz polluants).

6.2 Expériences sur moteur réel

Cette section présente les expériences menées sur un véritable moteur. Dans ce cadre, de nombreuses contraintes techniques viennent s'ajouter, notamment liées à la communication avec le matériel embarqué. Les conditions expérimentales sont décrites, avant de montrer les résultats obtenus.

6.2.1 Cadre expérimental

Les résultats présentés dans cette section sont issus de tests effectués sur un moteur essence monocylindre 125cm^3 de marque Peugeot. Celui-ci est instrumenté de manière à pouvoir mesurer des grandeurs, telles que des températures, des pressions, ou des concentrations de gaz, et les récupérer notamment via le boîtier de contrôle électronique (ECU). La liaison entre

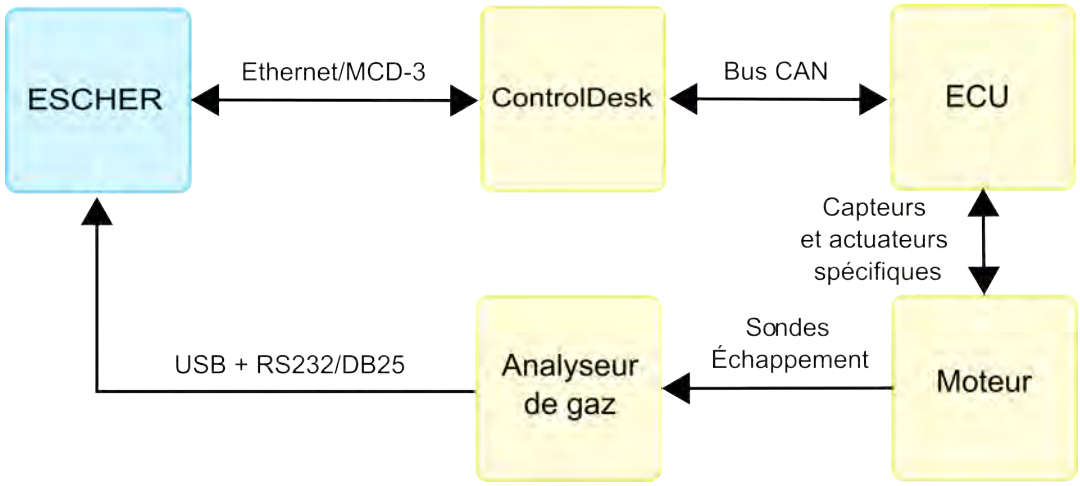

FIGURE 6.9 – Les différents systèmes impliqués lors des tests sur moteur réel.

les instruments du moteur et l'ECU est assurée grâce à divers supports filaires spécifiques (analogiques, fréquentielles, ou autres). Un bus CAN (*Controller Area Network*), un type de réseau très répandu dans le domaine automobile, permet la communication depuis l'extérieur avec l'ECU. Celle-ci se fait via un logiciel appelé ControlDesk, qui permet de lire le contenu de l'ECU (notamment les valeurs mesurées sur les capteurs), de calculer des grandeurs à partir des valeurs lues, et de modifier des paramètres (par exemple, commander l'avance à l'allumage). ESCHER est lui-même connecté à ControlDesk via un protocole de communication particulier, MCD-3 (pour *Measurement, Calibration, Diagnostics*), reposant sur Ethernet et permettant de lire et écrire des valeurs directement dans l'ECU. Enfin, un

analyseur de gaz est relié à l'échappement du moteur. Il détermine les concentrations de divers polluants (par exemple, le monoxyde de carbone) et envoie les données via une sortie série (RS232/DB25) interfacée avec le port USB du PC sur lequel tourne ESCHER. L'appareillage mis en œuvre est illustré par la figure 6.9.

6.2.1.1 Ajustements du système

Ces installations introduisent une contrainte technique importante : le délai de communication. En effet, si l'ECU est capable d'envoyer et de recevoir des données vers le moteur environ toutes les 10ms, la communication entre ESCHER et l'ECU prend, quant à elle, environ trois secondes. En outre, l'analyseur de gaz demande une quinzaine de secondes pour mettre à jour les données qu'il envoie. Pour contourner ce problème, un délai d'attente a été introduit dans ESCHER. Les agents attendent ainsi vingt secondes entre chacun de leurs cycles de vie.

Cela pose cependant un nouveau problème. En effet, les Agents Contrôleurs se retrouvent ainsi à avoir tous les mêmes perceptions, au même moment, et à agir en même temps (à l'échelle du procédé). Aussi, la résolution déterministe de la SNC 1 (en l'absence de propositions reçues, effectuer l'action opposée si la criticité augmente, garder la même action sinon) compromet l'exploration de l'espace d'états de l'environnement, et donc la convergence adéquate de ESCHER. Par exemple, si tous les Agents Contrôleurs sont en train d'augmenter la valeur de leur entrée, et que le niveau de criticité augmente, ils vont tous décider de descendre, en même temps. Or, il est probable que la bonne solution soit que seuls certains d'entre eux changent d'action. Cela vient du fait que les agents perçoivent, décident et agissent en même temps. C'est un problème équivalent à celui du *bar d'El Farol*, un problème issu de la théorie des jeux. L'unique solution connue à ce problème consiste à introduire des probabilités dans le raisonnement des agents. Aussi, nous avons modifié la résolution de la SNC 1. Plutôt que de systématiquement inverser son action si le niveau de criticité maximum augmente, un Agent Contrôleur en SNC 1 a une probabilité de $1 - \frac{1}{N}$ de ne pas modifier son entrée, où N est le nombre total d'Agents Contrôleurs dans le système.

Enfin, au moment d'effectuer ces expériences sur moteur réel, la SNC 9 (l'ajustement de l'amplitude de l'action) n'était pas finalisée. Aussi, cette dernière était-elle désactivée, et l'amplitude de l'action sur chaque entrée du moteur était un paramètre supplémentaire à définir. Pour le reste, le seul paramétrage d'ESCHER consiste à lui indiquer la référence des entrées qu'il doit contrôler, celle des sorties qu'il pouvait observer, et bien sûr lui fournir les fonctions de criticité adéquates.

6.2.1.2 Contrôle ou auto-calibration ?

Dans nos tests, le moteur est à chaque fois placé dans un point de fonctionnement donné. En d'autres termes, le régime et la charge moteur sont fixés et ne varient pas lors d'un même test. ESCHER contrôle alors des paramètres tels que le débit de carburant, ou l'avance à l'allumage, afin d'optimiser le fonctionnement du moteur. Ici, il ne s'agit donc pas de remplacer le rôle d'un contrôleur moteur classique qui transforme une demande de

couple en actions sur le moteur. En effet, les consignes portent sur des critères différents, et répondent, en fait, aux besoins de la calibration, les valeurs données par ESCHER étant directement utilisées par l'ECU pour calculer les commandes à effectuer sur le moteur (voir figure 1.5). Aussi, les expériences suivantes peuvent être considérées aussi bien comme le contrôle de l'ensemble ECU et moteur, que comme la calibration automatique d'un ECU pour l'optimisation du comportement du moteur.

6.2.2 Optimisation du couple

Les premiers essais se limitent à un petit nombre de paramètres à manipuler et de sorties à optimiser. Dans cet exemple, le moteur est placé à un régime de 5000 tours/min avec une charge de 870 mbar dans le collecteur d'admission. ESCHER contrôle la masse totale de carburant injectée à chaque injection, ainsi que l'avance à l'allumage, et a pour objectif de maximiser la pression moyenne indiquée (PMI), qui donne une mesure du couple développé par le moteur.

La masse de carburant injectée se mesure en milligrammes par injection (mg/cp), et l'avance à l'allumage en degré vilebrequin (°v, c'est-à-dire la position du piston dans le cylindre au moment du déclenchement de la combustion). La PMI, quant à elle, se mesure en bars. La PMI est une grandeur très instable, en particulier dans le cas d'un moteur monocylindre comme le nôtre. Travailler à haut régime et à forte charge (comme c'est le cas dans cette première expérience) permet de réduire son instabilité.

La fonction de criticité utilisée ici est strictement décroissante (car on veut maximiser la PMI). Puisque l'on ignore la PMI maximale, on est dans l'impossibilité de paramétrer la fonction de manière à ce qu'elle renvoie zéro lorsque le maximum de PMI est atteint. C'est pourquoi on ne s'attend pas à ce que la criticité soit nulle à la fin du test, mais seulement inférieure à son niveau de départ. Le fait que le niveau de criticité ne soit pas nul dans le cas idéal n'a qu'un impact visuel sur les courbes et n'influe en rien sur la convergence de ESCHER (celui-ci se basant uniquement sur les variations de niveau de criticité, et non sur leur valeur). Ces considérations sont valables pour toutes les fonctions de criticité utilisées lors des expériences effectuées sur le moteur.

La figure 6.10 montre l'évolution des entrées contrôlées par ESCHER, de la sortie optimisée, du nombre d'Agents Contextes et du niveau de criticité. À l'initialisation, la masse de carburant injectée est faible (7 mg/cp) en regard du point de fonctionnement courant. Le moteur risque fortement de caler si elle est diminuée. Bien sûr, ESCHER, qui n'a aucune connaissance sur le fonctionnement du moteur, ignore (pour le moment) ce fait. Aussi, sa première action est de diminuer chacun des paramètres. Cela entraîne une chute de la PMI, et donc une hausse dramatique de la criticité.

ESCHER trouve rapidement le moyen de faire diminuer le niveau de criticité, en augmentant d'abord la masse de carburant injectée, puis l'avance à l'allumage. La PMI finit par atteindre son maximum (environ 9 bar), la criticité ne diminue donc plus. ESCHER se stabilise alors à 11.50 mg/cp de carburant injecté, avec une avance de 24°v. La diminution des deux entrées au cycle 24, alors que le système est stable, s'explique par le bruit sur la

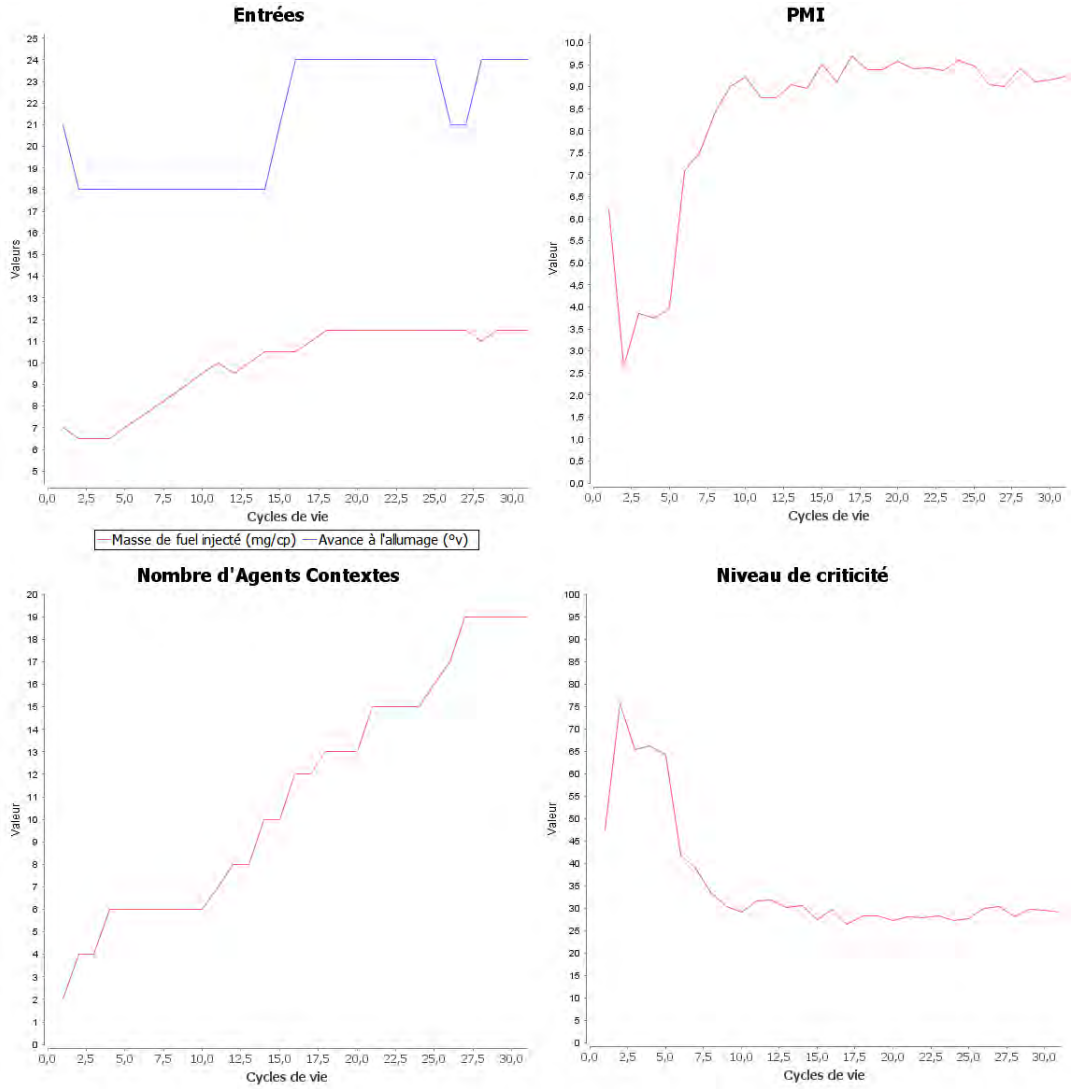

Figure 6.10 – Optimisation de la PMI par le contrôle de deux paramètres.

PMI qui engendre des variations du niveau de criticité. On note, cependant, que ESCHER se corrige rapidement.

L'analyseur de gaz n'intervenant pas dans cette expérience, la pause entre les cycles d'ESCHER est plus courte (3 secondes). ESCHER est parvenu à faire gagner 3 bar de PMI en 30 cycles, soit 90 secondes, plaçant cette dernière à son maximum pour le point de fonctionnement considéré. Le même résultat demande à un metteur au point expérimenté (et habitué au moteur étudié) environ vingt minutes avec les méthodes courantes.

6.2.3 Optimisation du couple et de la consommation avec seuils de pollution

Cette expérience se déroule en deux phases. Dans un premier temps, il s'agit de faire converger ESCHER vers un optimum, depuis un réglage quelconque. Dans un deuxième temps, le moteur est initialisé avec le réglage optimal précédemment obtenu. ESCHER est alors lancé avec les mêmes objectifs, mais sans avoir conservé la mémoire du test précédent (il n'y a donc, comme d'habitude, aucun Agent Contexte à l'initialisation). Nous voulons ainsi vérifier que ESCHER ne diverge pas si le minimum de criticité est atteint dès l'initialisation.

6.2.3.1 Convergence depuis un état quelconque

Pour ce test, le moteur est placé dans un nouveau point de fonctionnement (régime 2500 tr/min, et charge de 750 mbar). ESCHER contrôle à nouveau la masse de carburant injectée et l'avance à l'allumage, mais également le phasage de l'injection. Ce nouveau paramètre correspond au déclenchement de l'injection par rapport à la position du piston dans le cylindre, il est donc mesuré en degré vilebrequin. Il y a désormais quatre réponses à surveiller :

- la PMI doit toujours être optimisée ;
- il faut également minimiser la consommation spécifique, mesurée en g/kWh ;
- maintenir l'émission d'hydrocarbures (HC) sous un seuil de 500 ppm (parts par million) ;
- maintenir la proportion de monoxyde de carbone (CO) en-dessous de 3% à l'échappement.

Les trois derniers critères peuvent se montrer antinomiques avec le premier. En effet, le moyen le plus efficace pour augmenter la PMI est d'injecter plus de carburant. Cependant, cela a pour effet d'augmenter la consommation, ainsi que les émissions de polluants. Il faut alors jouer avec l'avance à l'allumage et le phasage de l'injection, afin de tirer le plus de puissance possible de la combustion. C'est ce que doit apprendre ESCHER.

La figure 6.11 montre l'évolution des paramètres contrôlés et des niveaux de criticité, tandis que la figure 6.12 illustre celle des sorties. C'est la criticité de la consommation qui est la plus forte au départ. ESCHER cherche donc à la diminuer en priorité. Il y parvient lors des 20 premiers cycles, notamment en augmentant l'avance à l'allumage de 10 à 26°v (par pas de 4°v), et en diminuant le phasage de l'injection de -150 à -400°v (par pas de 50°v), tandis que la masse de carburant injectée oscille entre 6 et 7 mg/cp (par pas de 0.50 mg/cp), avant d'augmenter légèrement. Cela a également pour effet d'améliorer la PMI.

Au dixième cycle, c'est la PMI qui devient la plus critique, cependant, les mêmes actions continuent d'être bénéfiques et sont poursuivies. Au vingtième cycle, toutefois, le seuil de CO est dépassé, et son niveau de criticité fait un bond. ESCHER tente alors de nouvelles actions pour remédier à ce problème. Il poursuit, par exemple, la baisse du phasage, et redescend l'avance à l'allumage. Cela va notamment provoquer un pic de consommation (et une baisse de PMI) entre les cycles 45 et 50, ainsi que de légers dépassements du seuil d'hydrocarbures (mais qui ne deviennent jamais plus critiques que les autres critères). Finalement, après

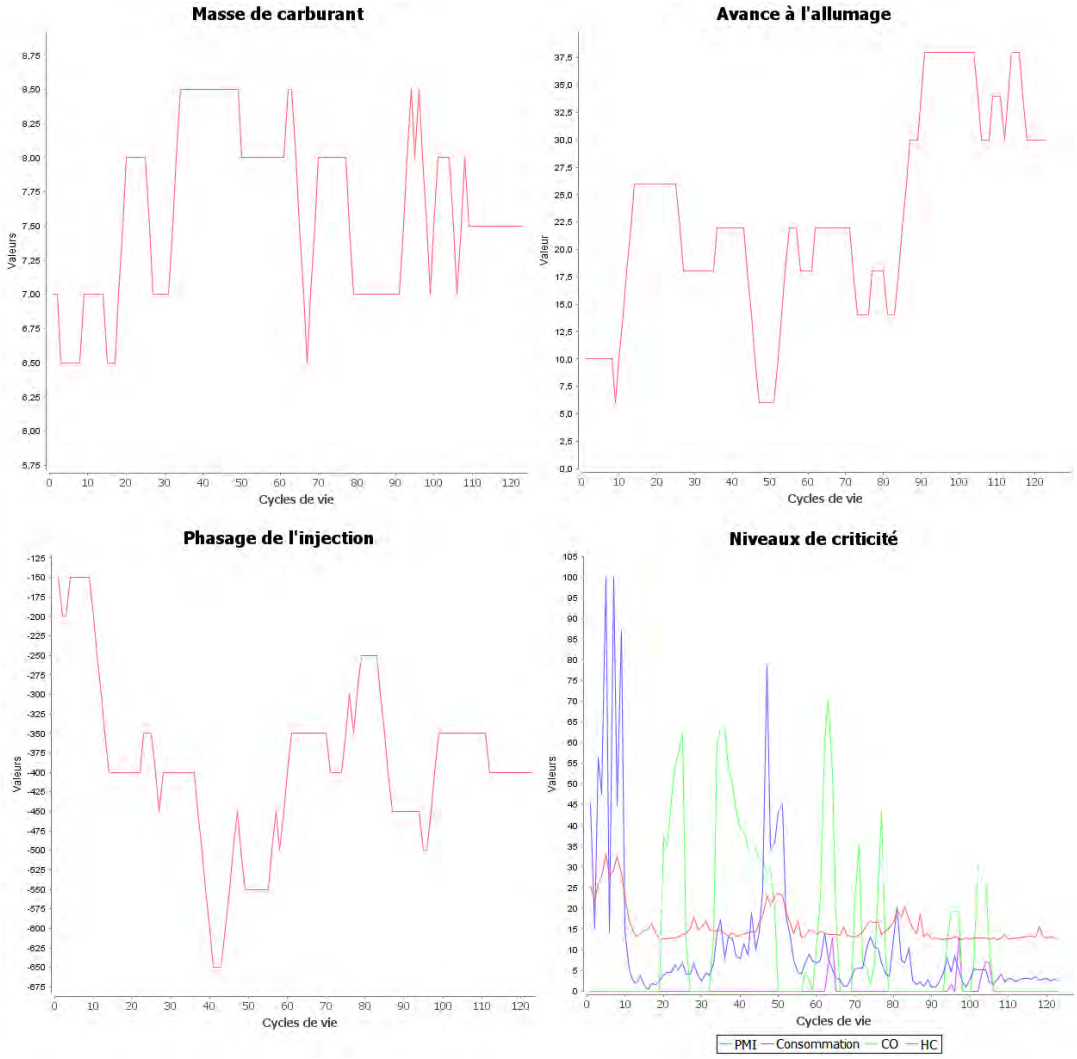

Figure 6.11 – Entrées et niveaux de criticité lors du contrôle de trois paramètres.

quelques oscillations, ESCHER parvient à faire en sorte que les deux seuils soient respectés, tout en maintenant une PMI haute et une consommation basse.

À la fin de l'expérience, la PMI est autour de 8 bar (soit 2 bar plus élevée qu'au départ), tandis que la consommation est d'environ 275 g/kWh (soit 165 g/kWh de moins). Les émissions de polluants sont quant à elles supérieures à leur valeur de départ, mais restent inférieures au seuil imposé. Le réglage conservé est le suivant :

– 7.50 mg/cp de masse de carburant injectée (contre 7 mg/cp au départ),
– 30°v d'avance à l'allumage (contre 10°v au départ),
– injections effectuées à -400°v (contre -150°v au départ).

Ces valeurs seront reprises comme valeurs initiales pour le prochain test.

Cet essai a duré 123 cycles de ESCHER, soit 41 minutes. Cela est environ deux fois plus rapide que ce que le même résultat demanderait à un expert humain.

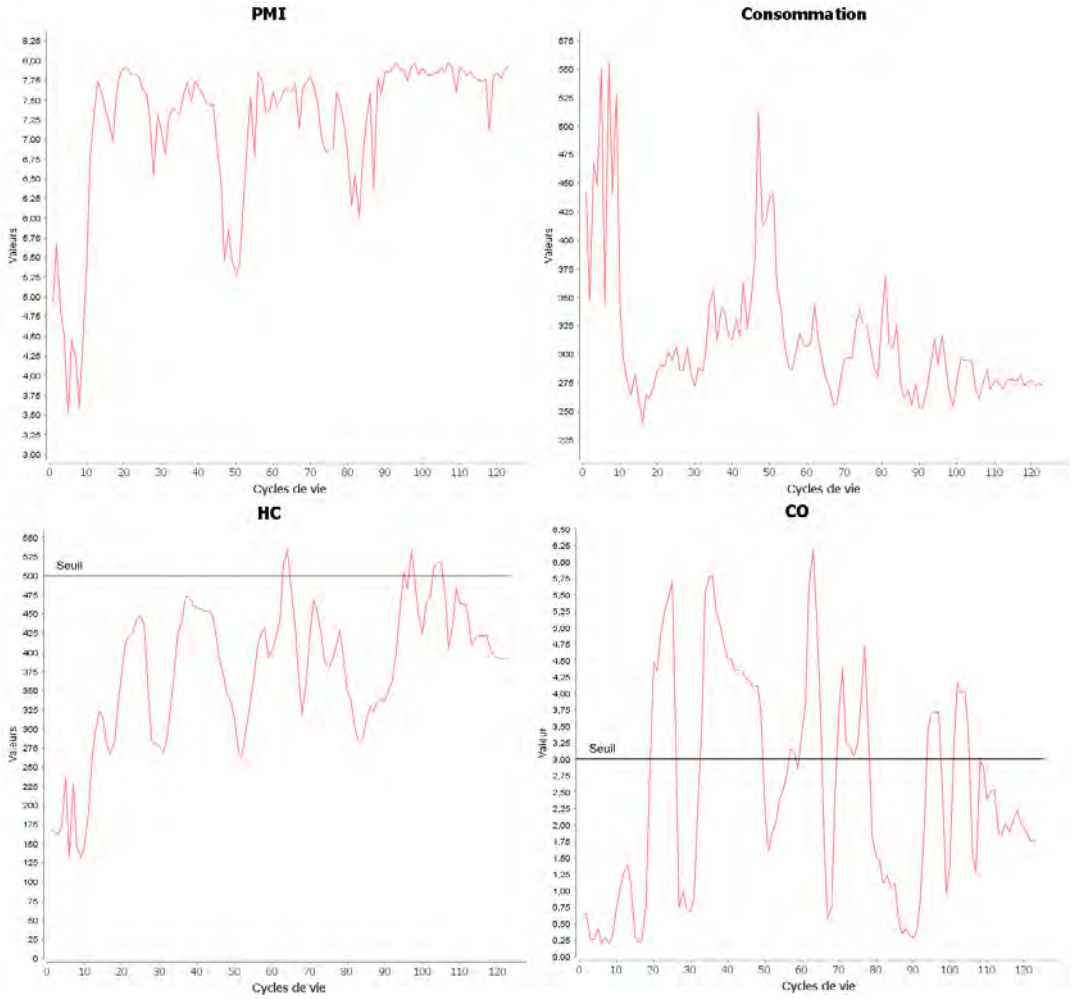

FIGURE 6.12 – Maximisation, minimisation, et seuils sur les sorties.

6.2.3.2 Maintien dans un état optimal

Ce second test vise à vérifier que ESCHER ne diverge pas si le moteur se trouve dans un état optimal dès le démarrage. Les paramètres sont donc initialisés avec les valeurs trouvées lors de l'essai précédent, et ESCHER est relancé (tous les Agents Contextes ont donc été supprimés).

La figure 6.13 montre les entrées et les niveaux de criticité, et la figure 6.14 les sorties. On observe que les entrées ne restent pas tout à fait stables. Cela est dû au bruit sur la PMI, dont le niveau de criticité est le plus haut. ESCHER essaye ainsi de corriger des augmentations dont il n'est pas responsable. Cependant, les variations restent très légères, et surtout, elles n'influencent pas significativement les sorties, et donc les niveaux de criticité.

En effet, au-delà d'un certain seuil, l'avance à l'allumage n'améliore plus le rendement de la combustion. Ainsi, l'augmentation effectuée par ESCHER, qui l'a fait varier de 30°v à un maximum de 42°v, est sans effet notable. La seule sortie à connaître des variations

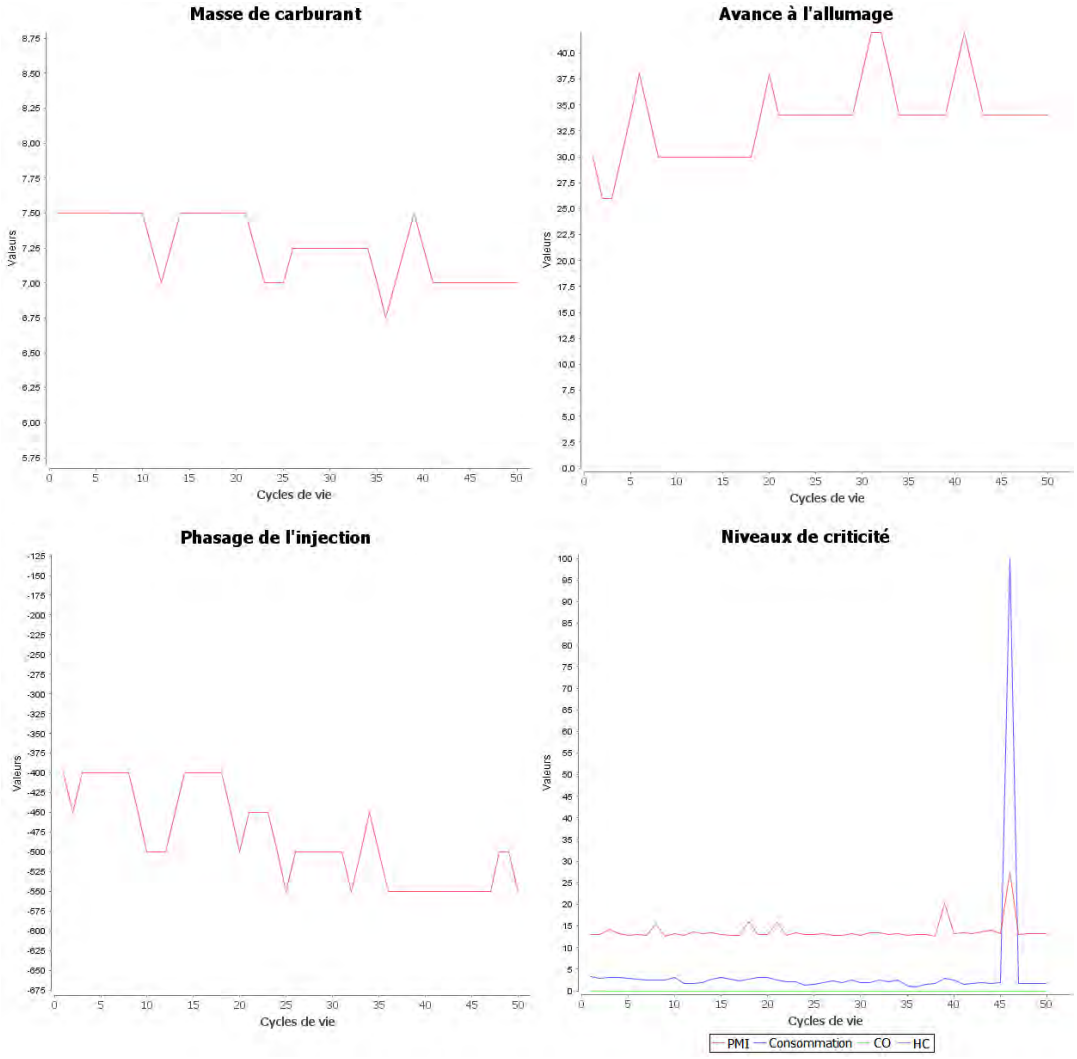

Figure 6.13 – Entrées et niveaux de criticité lorsque l'état initial du moteur est déjà optimal.

remarquables est la proportion de monoxyde de carbone, qui est fortement influencée par la masse de carburant injectée (qui elle reste relativement stable, oscillant entre 7 mg/cp et 7.50 mg/cp). Toutefois, les émissions de polluants restent constamment en dessous des seuils définis, les niveaux de criticité associés demeurent donc nuls.

Au cycle 46, on remarque un pic du niveau de criticité de la consommation, qui passe brusquement de 2 à 100, avant de redescendre à 2. Ce pic a, en fait, été provoqué par l'instabilité de la PMI, qui intervient dans le calcul de la consommation par ControlDesk. On voit que lors de ce même cycle, la PMI semble chuter de 7.75 bar à 4.5 bar. Cela provoque une valeur négative de consommation, ce qui est incohérent, et entraîne la valeur aberrante de niveau de criticité. Néanmoins, cela ne perturbe pas ESCHER, qui maintient efficacement le moteur dans son état optimal.

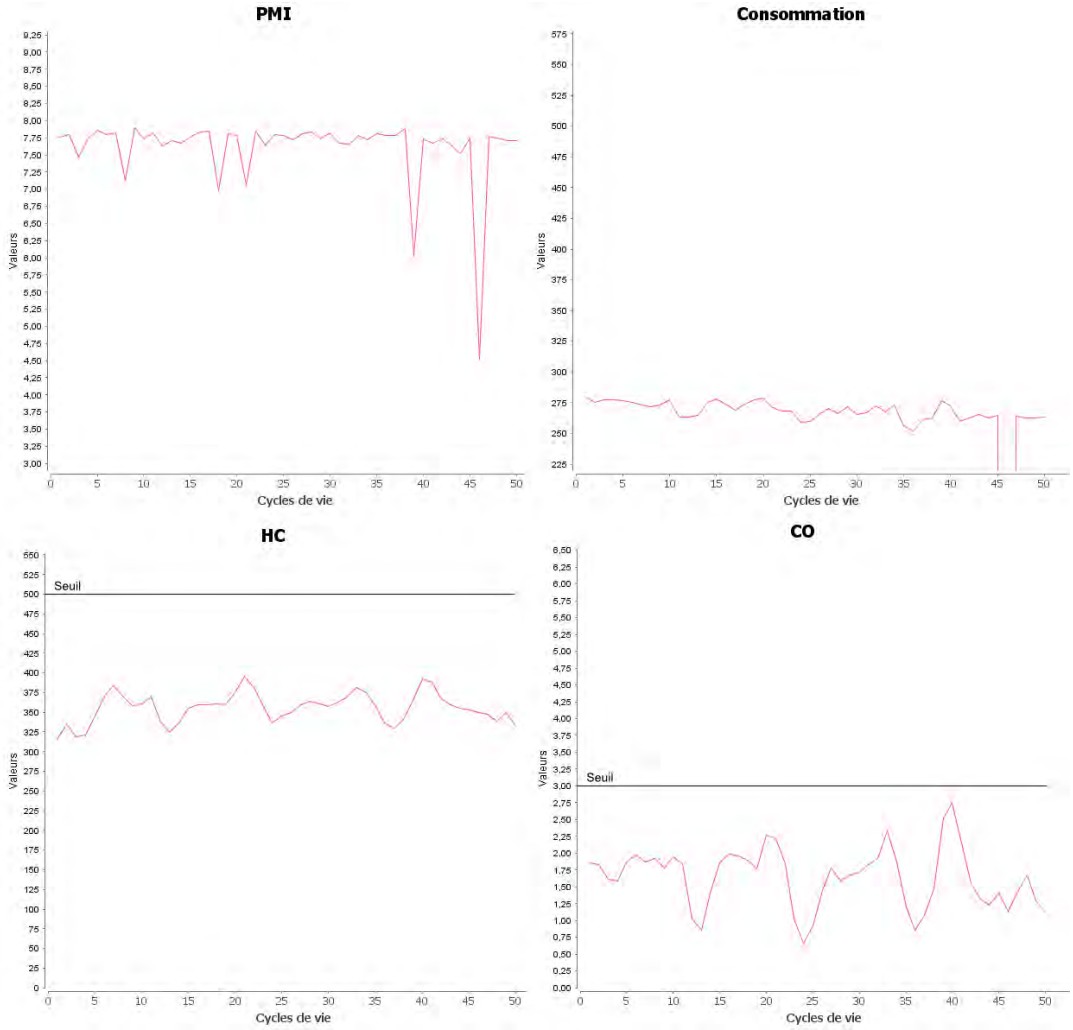

FIGURE 6.14 – Variations des sorties lorsque l'état initial du moteur est déjà optimal.

6.2.4 Un cas d'optimisation habituel

Ce nouveau test est un cas d'optimisation classique, conforme à ce qui se fait dans l'industrie. Il est similaire à l'essai précédent : les paramètres contrôlés et les sorties surveillées sont les mêmes, et le point de fonctionnement considéré (3500 tr/min, 800mbar) est proche du précédent. Les critères sur la PMI (maximiser) et sur la consommation (minimiser) sont également conservés. Les critères sur les émissions de HC et de CO sont quant à eux modifiés. Il ne s'agit plus de respecter un seuil, mais de minimiser ces deux sorties. En outre, le paramétrage de l'amplitude de l'action a été changé. Afin d'obtenir un réglage éventuellement plus fin, la masse de fuel injectée varie maintenant par pas de 0.25 mg/cp (au lieu de 0.50 mg/cp), et l'avance à l'allumage par pas de 2°v (au lieu de 4°v).

On observe, sur la figure 6.15, que le critère le plus critique en début d'expérience est celui de la PMI, suivi de près par celui de la consommation. Après une première erreur où il

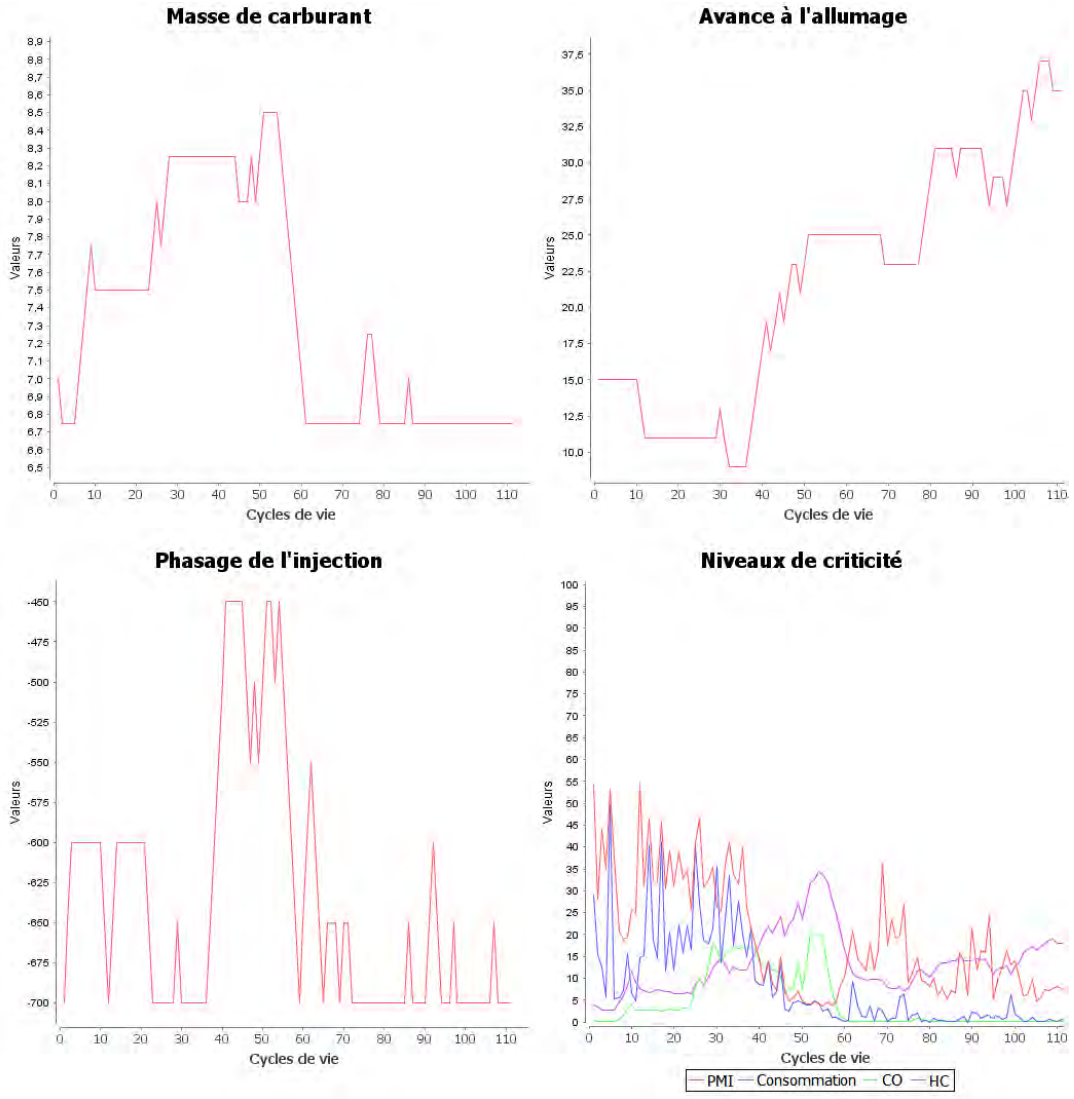

Figure 6.15 – Entrées et niveaux de criticité lors d'une optimisation classique.

diminue notamment la quantité de carburant injectée, ESCHER arrive finalement à diminuer les niveaux de criticité de la PMI et de la consommation. Il a pour cela augmenté la masse de carburant (de 7 mg/cp à 8.25 mg/cp), le phasage de l'injection (de -700°v à -450°v), et diminué l'avance à l'allumage (de 15°v à 9°v).

La PMI connaît ainsi une hausse significative (passe de 6 bar environ à près de 9 bar). Cependant, les actions de ESCHER ont également pour effet d'augmenter les émissions de polluants, en particulier celles d'hydrocarbures, dont le niveau de criticité devient le plus important à partir du cycle 40. Lors des cycles suivants, ESCHER comprend qu'il faut diminuer la quantité de carburant et le phasage de l'injection, tout en augmentant l'avance à l'allumage, pour diminuer la pollution et conserver tout de même une PMI acceptable.

À la fin de l'expérience, le niveau de criticité maximum est autour de 20, contre 50 au

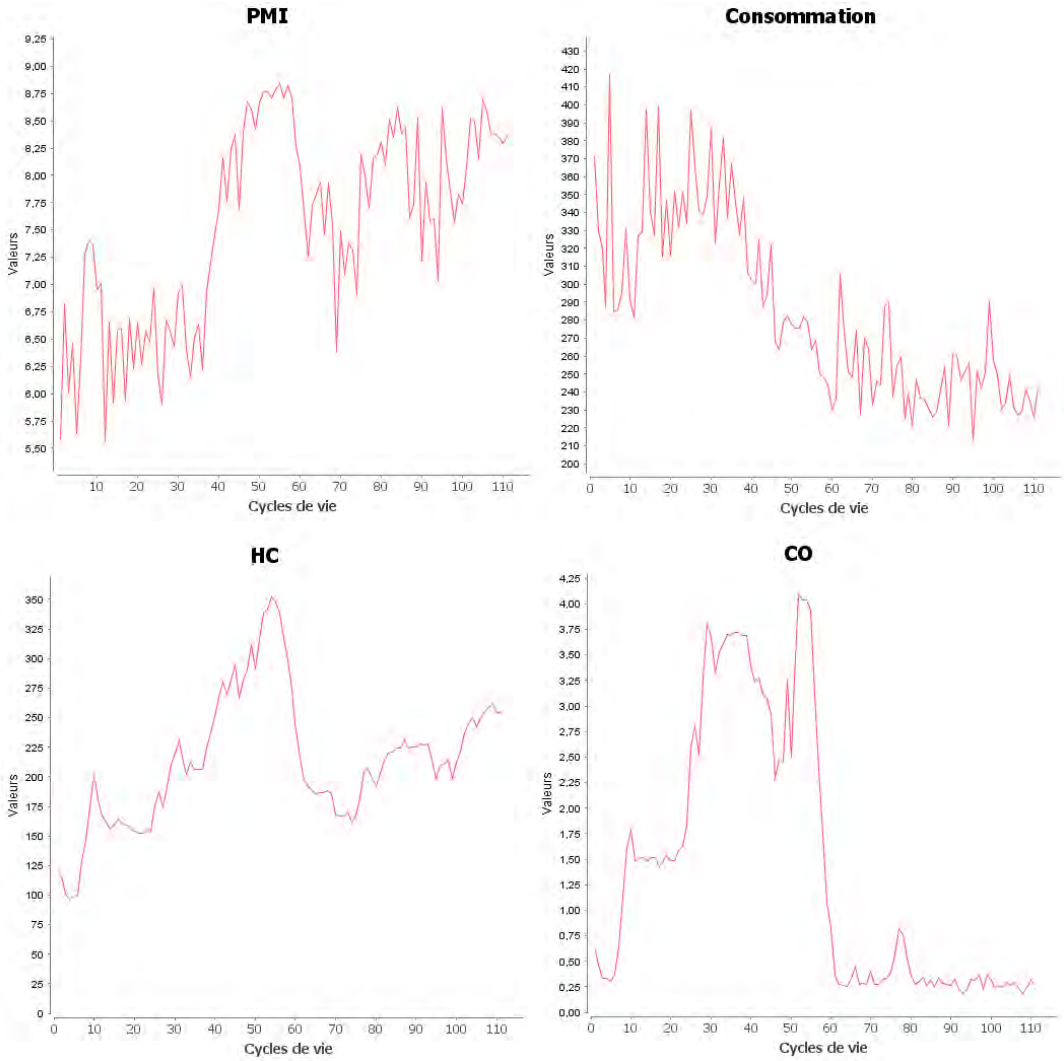

FIGURE 6.16 – Variations des sorties lors d'une optimisation classique.

départ. Conformément aux attentes, la PMI a augmenté, la consommation a diminué, et le monoxyde de carbone est resté à un niveau très bas. Seuls les hydrocarbures ont légèrement augmenté, mais restent à un niveau tout à fait acceptable. En raison du compromis avec la pollution, la PMI n'est toutefois pas à son maximum (atteint entre les cycles 50 et 60), mais en demeure proche. Le réglage atteint est équivalent à celui que l'on obtient par la méthode manuelle habituelle. ESCHER a néanmoins été, encore une fois, plus rapide, il lui a fallu 112 cycles (environ 37 minutes) pour atteindre ce résultat, soit deux fois moins que ce que demande une personne physique.

6.2.5 Un cas d'optimisation inhabituel

Cette dernière expérimentation est un essai inhabituel. Le moteur est placé dans le même point de fonctionnement que lors de l'essai précédent. Mais, s'il s'agit bien d'optimiser les niveaux de criticités, ceux-ci représentent cette fois-ci des critères insensés. En effet, on demande à ESCHER de maximiser toutes les sorties, y compris la consommation et les émissions de polluants. Comme pour les expériences précédentes, il y a un compromis à faire (augmenter la consommation et la pollution n'est pas nécessairement bénéfique pour la PMI). Le but est ici de voir comment réagit ESCHER lorsqu'il est confronté à des plages de fonctionnement du moteur habituellement évitées.

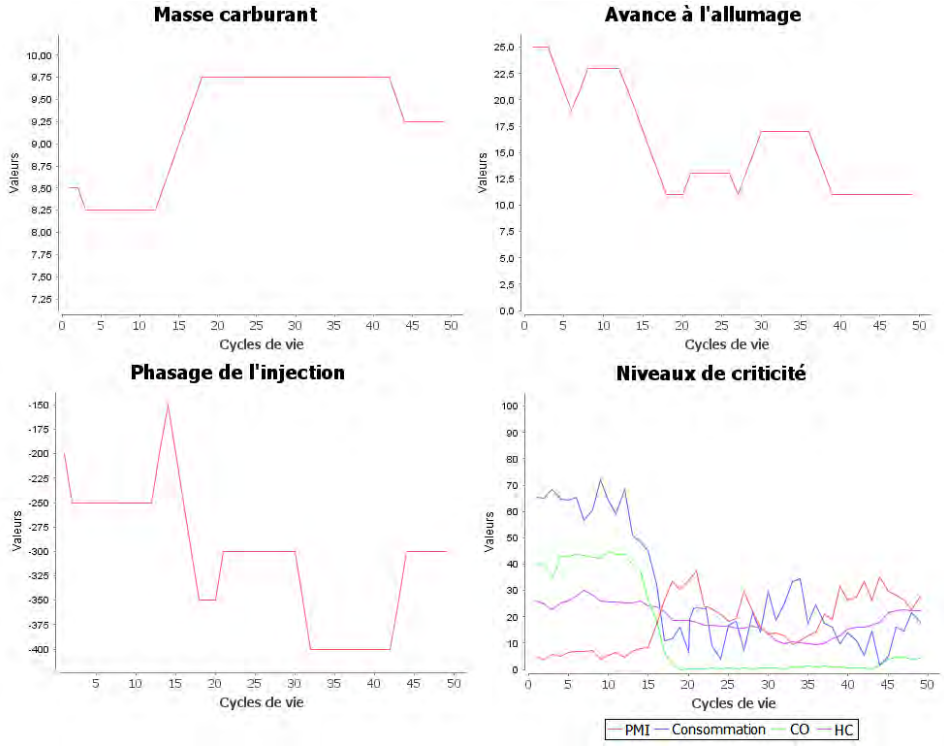

Figure 6.17 – Entrées et niveaux de criticité lors d'une optimisation inhabituelle.

La figure 6.17 montre les paramètres contrôlés et les niveaux de criticité, et la figure 6.18 les sorties à maximiser. C'est la consommation qui est au départ la plus critique. ESCHER cherche donc à la faire augmenter en priorité. Il diminue l'avance à l'allumage et le phasage de l'injection, et augmente la masse de carburant injectée. Cela a pour effet de fortement augmenter l'émission de CO et la consommation, diminuant du même coup leur niveau de criticité respectif. Cependant, cela a pour effet de diminuer la PMI, son niveau de criticité devient le plus important. En faisant varier le phasage de l'injection et l'avance à l'allumage, ESCHER fait osciller PMI et consommation, dont les niveaux de criticité sont tour à tour les plus hauts.

À la fin de l'expérience, le niveau maximum de criticité est passé de 65 à 28. La consom-

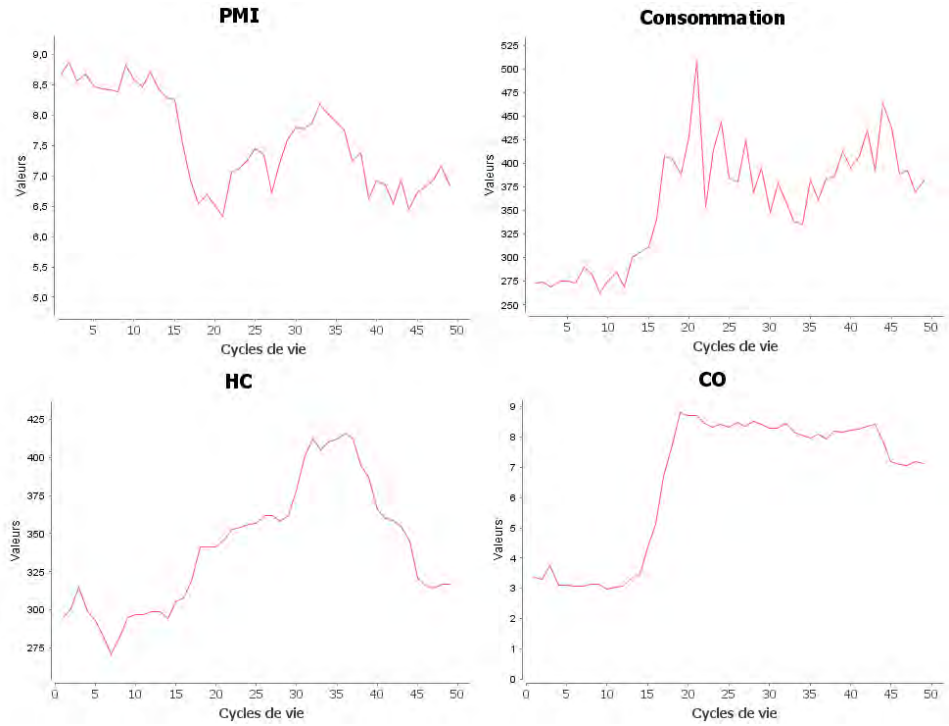

FIGURE 6.18 – Variations des sorties lors d'une optimisation inhabituelle.

mation et l'émission de monoxyde de carbone ont fortement augmenté, et l'émission d'hydrocarbures est légèrement supérieure à son niveau de départ. Cela s'est fait au prix d'une diminution de la PMI. ESCHER est parvenu à "optimiser" en 49 cycles, soit environ 16 minutes.

6.2.6 Bilan des expérimentations sur moteur

Cette section a présenté des exemples représentatifs des essais menés au banc moteur. Ils montrent l'applicabilité de ESCHER dans un cas réel, gérant divers critères de contrôle sur un système bruité et inconnu à l'avance.

La section suivante apporte un supplément de discussion quant à ces résultats, notamment en les comparant avec nos objectifs initiaux.

6.3 Discussion des résultats

Cette section discute les résultats obtenus vis-à-vis de nos objectifs initiaux, pour ESCHER comme pour BACH. Elle porte sur les points principaux suivants : le coût d'instanciation de ESCHER, son apprentissage et sa capacité à passer à l'échelle.

6.3.1 Validation de ESCHER

La conception de ESCHER s'est faite selon trois objectifs, énoncés dans les premiers chapitres de cette thèse. Nous voulions :

- un système de contrôle facile à appliquer à une instance particulière de procédé ;
- un système de contrôle capable de suivre l'évolution du procédé, autrement dit capable d'apprendre en parallèle du contrôle ;
- un système de contrôle capable de passer à l'échelle, c'est-à-dire de gérer simultanément un grand nombre de variables contrôlées et de critères d'optimisation.

Nous évaluons ici les résultats de notre système en regard de ces objectifs.

6.3.1.1 Faible coût d'instanciation

Le coût d'instanciation d'un système de contrôle à un procédé particulier provient principalement de deux éléments : la construction et l'utilisation d'un modèle analytique du procédé, et le paramétrage du contrôleur à proprement parler. Dans ESCHER, la solution proposée pour ces deux aspects est l'apprentissage, par l'auto-adaptation des agents.

Les expériences présentées dans ce chapitre montrent que ESCHER est capable de contrôler un système qui lui est a priori inconnu. Le contrôle est appris, sans modèle préalable du procédé, ce qui est un gain considérable en termes d'efforts à fournir pour son application. En outre, comme l'ont montré les expériences sur moteur réel, la convergence de ESCHER s'avère rapide.

De même, le paramétrage nécessaire pour faire fonctionner ESCHER est faible, puisque seules des informations basiques sur le procédé contrôlé sont nécessaires (le nombre d'entrées et de sorties, et leur référence). L'unique point délicat se situe au niveau des fonctions de criticité. Il n'est pas nécessaire d'ajuster celles-ci très finement, ESCHER est capable de suivre la moindre pente de criticité, quelle que soit son amplitude ou la valeur courante du niveau de criticité. Toutefois, si plusieurs critères sont à prendre en compte, il est important de bien réfléchir au compromis à faire, et de positionner les fonctions de criticité les unes par rapport aux autres en adéquation avec ce compromis. Dans le cas contraire, le problème risque d'être surcontraint, et ESCHER cherchera à converger vers des équilibres de paramètres qui seront nécessairement insatisfaisants pour l'utilisateur. Le principe des fonctions de criticité étant simple à appréhender, cela ne semble pas être une tâche complexe. Il serait néanmoins intéressant d'évaluer la complexité de cette tâche en procédant à des expérimentations poussées de mise en œuvre. On pourrait, par exemple, mesurer la difficulté éprouvée par plusieurs ingénieurs lors de l'application, au même système réel, de divers systèmes de contrôle intelligent (ESCHER, contrôleurs à base de réseaux de neurones flous, d'algorithmes génétiques, etc) qu'ils ne connaissent pas.

Ce premier objectif est donc atteint.

6.3.1.2 Apprendre en permanence

Afin de suivre l'évolution du comportement d'un procédé au cours du temps, le contrôle appris ne doit pas être figé. L'apprentissage doit se faire en continu. Cela permet également de s'adapter aux perturbations pouvant survenir.

Dans ESCHER, les agents sont en permanence en train de s'adapter, l'apprentissage n'est pas figé une fois que le système a convergé. Cela est notamment illustré par l'expérience présentée dans la section 6.1.5, dans laquelle ESCHER se réadapte après chaque perturbation du procédé.

Notons que cet apprentissage peut être facilement stoppé si un expert humain considère qu'il est suffisamment complet et ne désire plus d'adaptation. Il suffit de désactiver la détection de SNC, ce qui fige le corpus d'Agents Contextes. Le contrôle peut alors se poursuivre sans apprentissage.

Ce deuxième objectif est donc également rempli.

6.3.1.3 Passage à l'échelle

La plupart des approches de contrôle rencontrées lors de l'édification de l'état de l'art de ce document, notamment celles appliquées aux moteurs, se contentaient d'un faible nombre d'entrées et de sorties (trois ou moins). ESCHER a été testé avec succès sur des boîtes noires à 10 entrées et 10 sorties, c'est-à-dire 20 variables accompagnées de 10 consignes à respecter, ce qui est significativement supérieur aux techniques habituelles. Dans ces conditions, la durée de chaque cycle de ESCHER reste inférieur à une seconde, pour un total d'environ 4200 agents dans le système. Le temps de calcul n'est cependant la plus grosse préoccupation, et peut être amélioré par une optimisation de l'implémentation. C'est plutôt le nombre de cycles nécessaires à la convergence qui peut s'avérer problématique sur certains procédés réels. Toutefois, ce dernier ne semble pas nécessairement dépendre du nombre de paramètres contrôlés. On voit par exemple sur la figure 6.8 que certains tests à 10 entrées contrôlées convergent aussi vite que les cas précédents impliquant moins de variables.

Dans le cadre de cette thèse, ESCHER n'a pas pu être testé sur des procédés impliquant un nombre de variables de l'ordre de cent ou plus. Cela est en particulier dû à l'implémentation des boîtes noires générées qui, elle, ne passe pas à l'échelle. Cependant, la conception modulaire de ESCHER, avec des Agents Contrôleurs autonomes les uns par rapport aux autres, nous pousse à un optimisme raisonné quant à la capacité de ESCHER à passer à l'échelle sur le nombre de variables contrôlées et observables. La complexité induite ne devrait pas l'empêcher de trouver le minimum de criticité globale. Toutefois, cela reste encore à tester.

ESCHER est donc validé en regard des trois objectifs de cette thèse. Il présente cependant d'autres qualités intéressantes.

6.3.1.4 Robustesse au bruit

Une des inquiétudes avant de procéder aux tests sur moteur réel concernait le bruit sur les données. Par exemple, la PMI et la consommation sont particulièrement bruitées. La

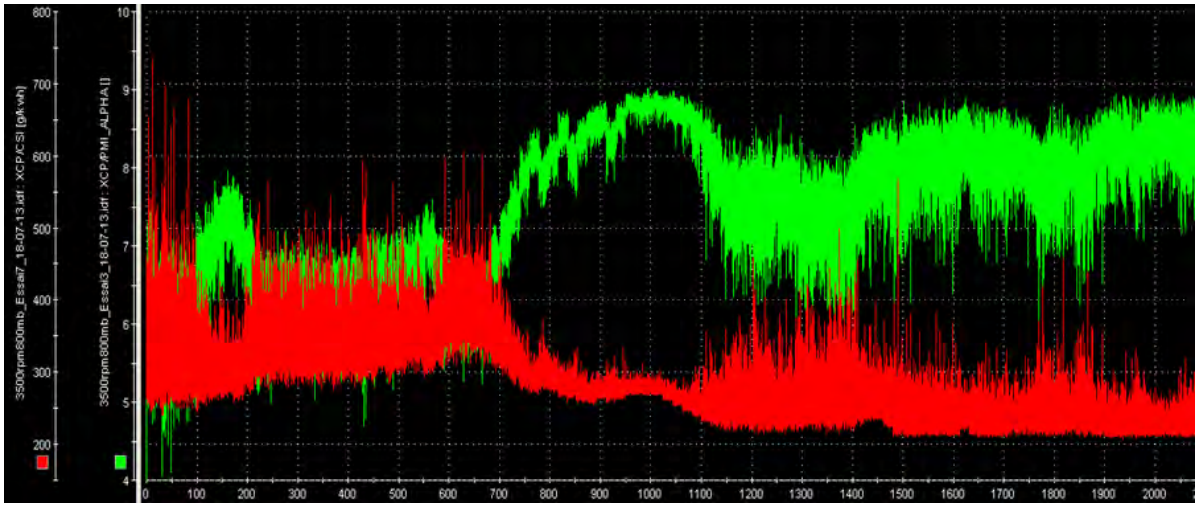

Figure 6.19 – PMI et consommation acquises par ControlDesk.

figure 6.19 montre ces deux sorties telles qu'acquises par ControlDesk (c'est-à-dire à une fréquence très élevée) lors de l'essai que nous avons présenté dans la section 6.2.4 (la PMI est en vert, et la consommation en rouge). ESCHER, qui est toujours parvenu à diminuer les niveaux de criticité, s'est révélé robuste à ce phénomène bien qu'aucun mécanisme ne prenne explicitement en compte la présence de bruit.

Cela provient de la combinaison de trois éléments : la forte réactivité de ESCHER pour corriger ses erreurs, le fait que celui-ci se base sur les variations de niveau de criticité (et non sur leur valeur courante), et le sous-échantillonnage des données perçues par les agents. En effet, la figure 6.19 montre les variables acquises par ControlDesk toutes les 10 ms, alors que dans le cadre de nos expériences, ESCHER ne communique avec ControlDesk que toutes les 20 secondes. Aussi, cela raréfie les occurrences d'erreur dans l'évaluation du sens de variation de la variable observée. Et lorsqu'une erreur survient, elle n'impacte que faiblement l'apprentissage de ESCHER qui est prompt à se corriger.

6.3.1.5 Généricité

Étant donné qu'aucune hypothèse restrictive n'a été faite quant au procédé contrôlé, il nous semble approprié de considérer ESCHER comme générique. En outre, l'abstraction que représentent les boîtes noires générées par BACH renforce cette opinion. Les boîtes noires peuvent s'apparenter à beaucoup de systèmes autres que les moteurs, et ESCHER est capable de les contrôler. Autrement dit, il est capable d'apprendre le contrôle de tout type de systèmes, du moment qu'il est possible d'observer la réaction des sorties en fonction des actions sur les entrées.

Cette contrainte est cependant plus forte qu'il n'y paraît, et exclut en fait les systèmes dont les effets d'une action se font sentir sur le très long terme. Ceci est une des limites de ESCHER, qui sont abordées dans la section suivante.

6.3.2 Limites et perspectives de ESCHER

L'approche AMAS suivie avec ESCHER est en rupture avec les méthodes classiques de contrôle. Cela permet de solutionner certains problèmes récurrents, mais pose également de nouvelles limites. Aussi, bien qu'ayant été validé en regard des objectifs initiaux, ESCHER laisse place à des améliorations.

6.3.2.1 Formalisation

La limite la plus importante concerne l'aspect théorique de l'approche. En effet, rien ne permet de prouver formellement que ESCHER converge systématiquement vers un optimum, ni que l'optimum trouvé est global. En outre, l'apprentissage a été conçu pour se poursuivre indéfiniment. Aussi, aucun critère ne permet de déterminer si le compromis actuel de criticité est le résultat final du contrôle, ou si ESCHER va finir par trouver une meilleure solution.

Ceci est illustré par la figure 6.20 qui montre l'évolution de la criticité maximale de 100 tests de 2000 cycles effectués avec la boîte noire à 4 entrées et 4 sorties utilisée dans la section 6.1.4, et initialisé de manière identique. Cette boîte noire est particulièrement difficile à contrôler, car l'ordre dans lequel les entrées sont modifiées impacte l'évolution du niveau de criticité. En effet, un certain nombre d'entrées n'influencent pas les niveaux de criticité tant qu'une autre entrée n'a pas atteint une certaine valeur. Ainsi, lorsque la criticité maximale arrête de diminuer, ESCHER doit trouver quelles sont les prochaines entrées qui doivent être modifiées, et cela peut prendre du temps. C'est pour cela que l'on observe des paliers de

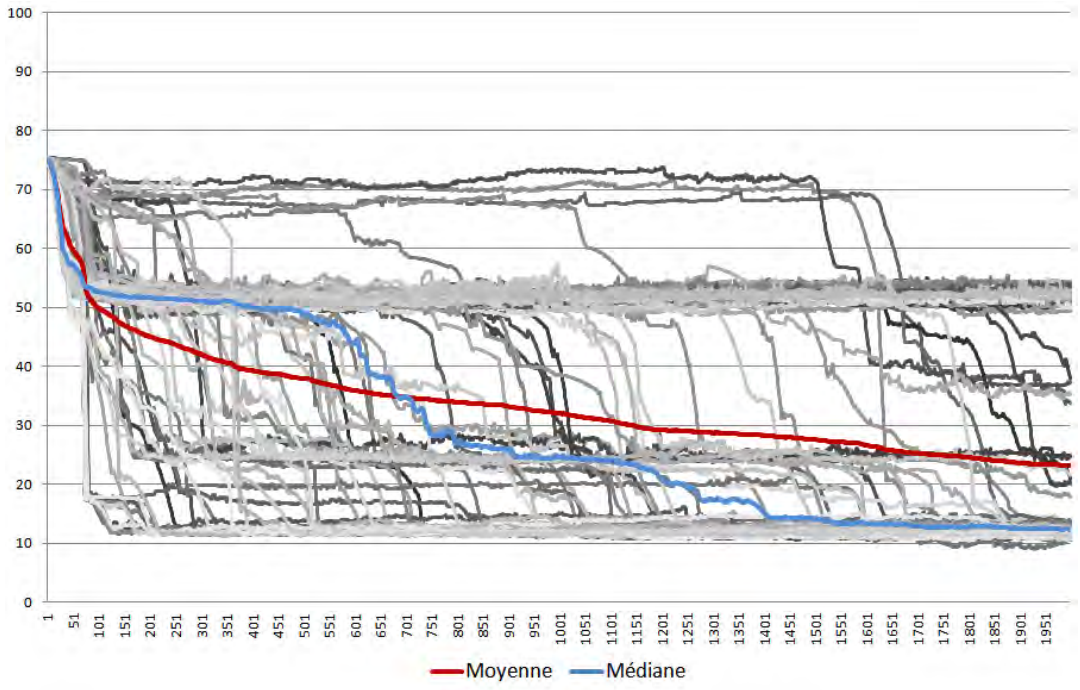

FIGURE 6.20 – Niveau de criticité maximale de 100 tests sur une boîte à 4 entrées et 4 sorties.

criticité sur la figure 6.20 (autour de 70, de 52, et de 25). On voit que, selon les tests, le niveau de criticité maximal met plus ou moins longtemps à passer d'un palier à l'autre. Certains tests atteignent la criticité minimale en moins de 100 cycles, tandis que d'autres n'y sont toujours pas au bout de 2000. Le palier autour de 52 semble particulièrement difficile à passer. La moyenne (en rouge), et surtout la médiane (en bleu), nous indiquent cependant que plus le temps passe, plus la proportion de tests à atteindre le palier de criticité minimale augmente. En prolongeant suffisamment le test, tous les niveaux de criticité devraient atteindre le palier le plus bas.

Toutefois, il est impossible de s'assurer, sans "ouvrir" la boîte noire, que le palier minimal de criticité atteint est effectivement le meilleur compromis possible. Rien ne dit qu'aucune amélioration du niveau de criticité (aller en-dessous du palier à 10 dans notre exemple) n'est possible en continuant le test sur une plus longue durée. Surveiller le nombre d'Agents Contextes créés peut être un bon indicateur, mais n'est pas suffisant. Lorsqu'il n'augmente plus, cela traduit une certaine stabilité du système, mais qui peut n'être que temporaire.

Ce problème concerne spécifiquement l'aspect apprentissage, une fois les Agents Contextes créés et stabilisés, ESCHER n'est plus gêné par ces paliers. La boîte noire utilisée pour cet exemple est un cas extrême où les paliers sont nombreux. Ce problème est par exemple beaucoup moins présent avec le moteur réel, mais impose tout de même la présence d'un superviseur humain qui décide du moment où l'essai peut être stoppé.

Deux pistes doivent être explorées pour améliorer cet aspect de notre système. D'une part, il manque probablement une SNC à découvrir qui permettrait aux Agents Contrôleurs de sortir plus rapidement d'un palier. D'autre part, formaliser l'approche AMAS est une tâche extrêmement difficile, une perspective à long terme, qui permettrait de prouver des propriétés telle que la convergence, et ainsi nous aider dans l'analyse des résultats obtenus.

6.3.2.2 Bruit et latence

Si le bruit a finalement eu un impact limité sur les expériences menées sur moteur réel, la raison de ce fait (sous-échantillonner la variable) n'est pas satisfaisante. En outre, la SNC 9 (l'apprentissage de l'amplitude d'une action) n'était pas implémentée lors de ces tests. Or, elle se base sur la vitesse de variation des niveaux de criticité, qui peut être fortement impactée par le bruit. Cela laisse à penser que la résolution de cette SNC ne fonctionnerait pas dans les conditions de nos expériences.

ESCHER est incapable de gérer les cas où l'effet d'une action sur une entrée ne devient observable sur une sortie qu'après une longue durée. Par exemple, avec un moteur, il faut modifier certains paramètres et les maintenir dans leur nouvel état pendant un temps prolongé (de l'ordre de la centaine de cycles pour ESCHER) avant que la température ne commence à diminuer. ESCHER a peu de chance de voir des Agents Contextes rendre compte de ce fait, surtout si d'autres variables ont changé de valeur entre temps.

Ces deux aspects, le bruit et la latence, sont des éléments propres à l'environnement de ESCHER qui devraient être appris par ce dernier. Les prochains travaux doivent se concentrer sur la découverte des SNC permettant de gérer le bruit et la latence. Nous pensons que

les SNC correspondant à ces deux problèmes sont en fait identiques. En effet, il s'agit de déterminer si une variation perçue en sortie est due au bruit, ou bien à une action passée.

6.3.2.3 Extraire les connaissances acquises

ESCHER acquiert des connaissances au sujet du procédé contrôlé au cours du temps. Ces connaissances sont distribuées dans tous les Agents Contextes, et ne sont donc pas accessibles directement depuis l'extérieur. Ceci n'est pas une limite en regard de la fonction première de ESCHER. Toutefois, il est dommage de ne pas pouvoir réutiliser ces connaissances à d'autres fins (comme par exemple obtenir un modèle calculable du procédé contrôlé).

Aussi, extraire les connaissances apprises par ESCHER représente une perspective à court terme intéressante. À première vue, la tâche semble relativement aisée, puisque les informations sont stockées explicitement dans les agents. Elle présente néanmoins quelques défis (notamment l'agrégation d'informations contradictoires provenant d'agents différents).

6.3.2.4 Le paradoxe du contrôle d'un système inconnu

ESCHER est un contrôleur capable de gérer un système préalablement inconnu (à l'exception de ses entrées et sorties). La fonction d'un contrôleur est de placer le système contrôlé dans un état désiré, spécifié par l'utilisateur. Mais, si le système contrôlé est véritablement inconnu, comment l'utilisateur peut-il être sûr de ne donner au contrôleur que des objectifs réalisables ?

Cela rejoint, en partie, le problème de savoir détecter si l'optimum a été trouvé ou si une meilleure solution va être découverte. C'est une limite qui ne provient pas du contrôleur lui-même, mais de la connaissance que l'utilisateur a du système contrôlé. Les capacités d'apprentissage d'un contrôleur permettent de contrôler un système qui est inconnu *du contrôleur*, mais l'utilisateur doit au minimum avoir une idée de ce qu'il est possible de faire ou non. L'apprentissage est un moyen d'épargner le travail que demande le transfert des connaissances de l'utilisateur au contrôleur (qui se fait habituellement par la construction d'un modèle mathématique, ou par un paramétrage important et précis).

Mais il est possible d'aller plus loin dans l'apprentissage. Un contrôleur pourrait, par exemple, être capable d'apprendre les limites du système qu'il contrôle, et ainsi avertir l'utilisateur en cas de consignes irréalisables. Le contrôleur apprenant pourrait également être capable de détecter ses propres limites actuelles, et d'apprendre comment les dépasser. Par exemple, ESCHER pourrait se rendre compte qu'il n'est pas capable d'apprendre les effets se produisant sur une longue échelle de temps. Dans le cas d'un AMAS, cela signifierait que le système est capable de détecter de nouvelles SNC (et de trouver comment les résoudre) qui n'auraient pas été prévues par le concepteur, voire de détecter des fonctions locales manquantes et de créer le type d'agent pouvant les accomplir.

Ce sont là des perspectives extrêmement stimulantes.

6.3.3 Validation de BACH

L'objectif avec BACH était de disposer d'une palette de cas de test, sous forme de boîtes noires, permettant d'éprouver ESCHER durant son développement en s'épargnant la lourdeur des expériences sur moteur réel. Il y avait néanmoins le risque que les boîtes noires générées par BACH soient une mauvaise abstraction, c'est-à-dire qu'elles présentent un comportement trop différent d'un moteur réel pour que les enseignements tirés de leur utilisation soient pertinents. Un moyen de vérifier la validité de ces boîtes noires était de poursuivre le développement de ESCHER jusqu'à son terme, et de le tester sur le vrai moteur. Si les performances de ESCHER sur chacun des systèmes sont équivalentes (sans avoir à le modifier profondément), alors BACH est validé.

Les seuls ajustements apportés à ESCHER ont été induits par le protocole de communication particulier avec le moteur. Pour le reste, le comportement de notre contrôleur sur le moteur s'est avéré tout à fait conforme à nos attentes, fondées sur les essais avec les boîtes noires générées. Aussi, nous considérons BACH comme validé dans le cas précis de l'évaluation de ESCHER pour son utilisation future sur un moteur à combustion.

6.3.4 Limites et perspectives de BACH

La question subsiste de savoir si BACH est également pertinent dans le cadre du développement d'autres systèmes d'apprentissage du contrôle, destinés éventuellement à d'autres systèmes à contrôler. En outre, il se peut que les capacités d'adaptation de ESCHER soient suffisamment poussées pour masquer des aspects difficiles du comportement du moteur réel qui ne sont pas présents chez les boîtes noires (ou inversement).

Aussi, un travail important à faire est de concevoir et d'effectuer des tests précis, visant à déterminer exactement quelles sont les différences fondamentales (s'il y en a) entre le comportement des entrées et sorties des boîtes noires générées et celles de systèmes réels. Cela rejoint le domaine de l'étude des systèmes complexes.

En outre, à partir d'un certain nombre d'entrées et sorties (environ une douzaine de chaque), les boîtes noires générées sont souvent inextricables. C'est-à-dire que les sorties ne varient quasiment pas quelles que soient les valeurs des entrées. Les phases d'auto-assemblage et d'auto-ajustement du processus de génération demandent donc à être affinées. Elles gagneraient certainement se dérouler simultanément, avec des ajustements effectués à chaque nouveau lien.

6.4 Bilan

Ce dernier chapitre a présenté les expériences conduites sur des boîtes noires générées par BACH, puis sur un moteur à combustion. Ces expériences ont permis de valider ESCHER selon les trois objectifs de cette thèse : la facilité d'instanciation, la capacité d'adaptation et celle de passer à l'échelle. ESCHER s'est également révélé relativement robuste au bruit, et présente tous les atouts pour être un système de contrôle générique.

Toutefois, ces résultats très encourageants sont à pondérer par les limites de ESCHER, qui sont également exposées dans ce chapitre. En particulier, le manque de recul théorique parait être le point le plus délicat. Si la validité de notre approche générale est assurée par la théorie des AMAS, sa concrétisation sous la forme de ESCHER n'offre pas, pour le moment, de garantie théorique.

Table 6.1 – Tableau comparatif des méthodes de contrôle et de ESCHER.

Critère	PID	Contrôle adaptatif	Contrôle intelligent	ESCHER
Généricité	+	+	+ +	+ +
Instanciation	- -	- -	- -	+
Adaptativité	- -	+	+ +	+ +
Apprentissage	Aucun (la connaissance du procédé est implicitement contenue dans le paramétrage	Limité (ajustement de paramètres d'une structure fixe)	Variable (de limité à très important)	Poussé (apprend directement le contrôle)

Le tableau 6.1 reprend le bilan du chapitre 2 auquel il ajoute ESCHER. Il est toutefois à prendre avec précaution puisqu'il compare un système particulier avec des approches générales qui regroupent des techniques hétérogènes.

Enfin, nos expériences ont également permis de valider la démarche entreprise avec BACH, à savoir utiliser des boîtes noires abstraites pour mettre au point un système d'apprentissage du contrôle. Ces boîtes noires ont permis de s'affranchir des contraintes techniques liées au banc moteur pour les nombreux tests jalonnant le développement de ESCHER.

Conclusion générale

Le contrôle de systèmes complexes, point de départ de cette thèse, est un domaine riche, trouvant ses racines dans de nombreux champs d'application, et dont les notions de base sont présentées dans le premier chapitre. Son étude nous a rapidement menés vers l'apprentissage artificiel, une discipline au cœur des approches modernes de contrôle. Aussi, le second chapitre est un état de l'art couvrant un vaste éventail de techniques de contrôle et d'apprentissage. Le plus gros défi actuel concerne l'application, à des cas réels, des techniques dites intelligentes. Bien souvent, les algorithmes mis en place sont incapables de franchir la fameuse "barrière de complexité", qui apparaît lorsque le problème (le système à contrôler, ou l'objet à apprendre) possède une dynamique non-linéaire et fait intervenir un grand nombre de variables. Les travaux les plus récents présentés dans l'état de l'art laissent à penser que la solution se trouve, en partie, dans la distribution du contrôle. Ainsi, les systèmes multi-agents, naturellement distribués, et en particulier les systèmes multi-agents adaptatifs et leur conception entièrement *bottom-up*, représentent une alternative séduisante. Dans de tels systèmes, l'auto-organisation coopérative des agents leur permet de trouver eux-mêmes, collectivement, une solution au problème qui leur est présenté. Ils sont abordés en détail dans le chapitre 3.

Le chapitre 4 introduit ESCHER, le système multi-agent adaptatif conçu et développé au cours de cette thèse afin de répondre au défi du contrôle de systèmes complexes, et notamment celui des moteurs à combustion. Dans ce système, les agents ignorent la tâche globale que le collectif doit effectuer. Chacun se concentre sur son propre but local, tout en gardant un comportement coopératif pour résoudre les problèmes (locaux) qui se présentent à lui. Ces résolutions locales provoquent des changements au niveau des agents (ajustements de paramètres, réorganisation des interactions, création ou suppression d'agents) qui se répercutent sur le comportement global de ESCHER, le guidant vers l'adéquation fonctionnelle.

Deux types d'agents en particulier sont au centre de l'activité du système : les Agents Contrôleurs et les Agents Contextes. Les premiers sont majoritairement responsables du contrôle proprement dit, tandis que les seconds apprennent les réactions de l'environnement. Les Agents Contrôleurs se basent sur les indications des Agents Contextes pour effectuer des actions, tandis que l'apprentissage des Agents Contextes dépend directement des choix que font les Agents Contrôleurs. Ainsi, apprentissage et contrôle sont couplés. Ils se construisent mutuellement, à l'image des *Mains se dessinant* (figure 1) de l'artiste hollandais dont notre

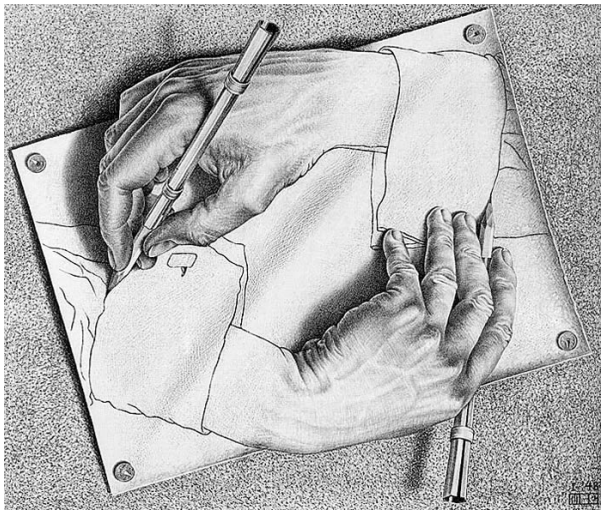

Figure 1 – Mains se dessinant (lithographie de M.C. Escher, 1948).

système emprunte le nom, Maurits Cornelis Escher. D'autre part, le pavage de l'espace, qui est un des thèmes récurrents des œuvres de M.C. Escher, est un aspect important de notre système. En effet, aucun agent ne centralise toute la connaissance acquise par ESCHER, celle-ci est distribuée parmi eux. Chaque Agent Contexte représente une portion de l'espace d'états de l'environnement pour laquelle les effets d'une action donnée sont connus. Pris isolément, un Agent Contexte n'est pas intéressant, c'est la "mosaïque" composée de tous les Agents Contextes qui l'est. Le fait que chacun des éléments de la "mosaïque" est capable de s'ajuster de manière autonome, afin de garder un tout cohérent, donne au système toute sa capacité d'apprentissage et d'adaptation.

Le développement de ESCHER a été grandement facilité par un autre système multi-agent adaptatif, également produit durant cette thèse, et nommé BACH. Présenté dans le chapitre 5, ce système a pour but de générer des boîtes noires faisant office de systèmes à contrôler pour ESCHER. Ces boîtes noires ont permis de procéder à de nombreux tests sans subir les lourdes contraintes techniques des expérimentations sur un moteur réel. Finalement, le chapitre 6 montre et discute les expériences effectuées, avec des boîtes noires générées mais également sur un vrai moteur. Les résultats obtenus ont permis de valider nos deux systèmes en regard de nos objectifs initiaux, tout en faisant apparaître certaines limites, laissant ainsi la place à de nombreuses perspectives.

Contributions

ESCHER constitue la principale contribution de cette thèse. Il offre des apports aussi bien dans le champ général du contrôle, que dans les domaines plus spécifiques de la calibration d'ECU et de l'étude des AMAS.

Au contrôle, il apporte un système capable de gérer simultanément plusieurs entrées

et sorties d'un système non-linéaire, sur lesquelles plusieurs consignes (éventuellement contradictoires) peuvent être appliquées. Il est générique, capable de passer à l'échelle, et ne nécessite pas de modèle préétabli du système contrôlé, tout passe par l'apprentissage. Cet aspect lui donne une grande facilité d'utilisation, et permet son instanciation rapide à un procédé donné. La réunion de tous ces critères est, à ma connaissance, inédite.

À la calibration d'ECU, il apporte un moyen d'automatisation, permettant un gain significatif de temps. En effet, les tests effectués sur moteur réel ont montré que ESCHER est capable d'amener le moteur dans un état optimal environ deux fois plus vite qu'un expert humain avec la méthode classique utilisée dans l'industrie. La supervision humaine actuellement nécessaire est très légère, et permet à l'ingénieur d'effectuer d'autres tâches en parallèle. En outre, utilisé comme outil d'auto-calibration, ESCHER peut s'insérer facilement dans le processus de développement d'un ECU, sans remettre en cause toute la chaîne de production.

Aux AMAS, il apporte une validation supplémentaire de l'approche, avec un système éprouvé sur des cas réels, hors simulation. Son développement a également permis de raffiner certains outils, comme les fonctions de criticité, l'utilisation d'AVT pour estimer des intervalles de valeur, ou encore produit des composants réutilisables via MAY. Enfin, il peut être vu comme une version largement améliorée du système Obsidian (Videau 2011), entre autres avec l'ajout des Agents Critères (qui permettent de gérer facilement plusieurs consignes pouvant chacune faire intervenir plusieurs variables), et celui du mécanisme de consigne dynamique (qui permet d'éviter de rendre obsolètes tous les Agents Contextes à chaque changement de consigne, et de conserver l'apprentissage).

Enfin, BACH est une contribution à part entière. Son apport va au-delà du cadre multi-agent et se situe au niveau du développement de systèmes capables d'apprentissage. La génération automatique produit des boîtes à la dynamique variée, utilisables dans tous les domaines où une approche "boîte noire" est possible, ce qui est, par exemple, le cas de la majorité des techniques d'apprentissage par renforcement, et de bon nombre d'algorithmes d'optimisation.

Perspectives

Les travaux présentés dans cette thèse ouvrent de nombreuses perspectives à court terme, principalement d'un point de vue applicatif, mais nourrissent également des projets à long terme.

À court et moyen terme

Plusieurs améliorations peuvent être apportées à ESCHER pour pousser l'apprentissage encore plus loin. En premier lieu, le bruit sur les données, et la latence entre une action sur les entrées et l'observation d'effets sur les sorties doivent être appris, afin de rendre le système plus robuste.

De même, certains paramètres, tels que la taille minimale des plages de validité, ou bien les coefficients des AVT et leur pas minimal, pourraient être appris pendant l'exécution. Il

serait même possible d'ajuster automatiquement les fonctions de criticité de manière à ce que leur définition puisse être vague. Ce serait alors à ESCHER de faire en sorte que tous les niveaux de criticité soient pris en compte, et à proposer leur équilibration. Cela rejoint certains travaux en cours sur les AMAS, qui cherchent à produire des systèmes capables de trouver leurs propres fonctions de criticité et de découvrir des SNC non prévues par le concepteur (Mefteh et al. 2013).

Conçu avant tout pour le contrôle, ESCHER doit également apprendre et, pour cela, explorer efficacement l'espace d'états de son environnement. Ce fait le rapproche des algorithmes d'optimisation. Aussi, son utilisation en temps qu'outil d'auto-calibration s'est révélée prometteuse. C'est pourquoi il est envisagé de produire une version alternative du système, dédiée entièrement à l'optimisation multi-critère. Cela implique notamment la définition d'un critère d'arrêt de l'exploration, afin que cette tâche n'incombe pas à un superviseur humain. Dans ce cas, la définition de nouvelles SNC, à même de répondre au problème des paliers de criticité, devient cruciale et doit être traitée en priorité. C'est cette voie qui semble susciter le plus d'intérêt de la part des industriels, et qui devrait donc être poursuivie dans un avenir proche.

À long terme

Les Agents Contextes ont été introduits pour la première fois dans Obsidian (Videau 2011), un système de contrôle de bioprocédés, puis réutilisés (et améliorés) dans trois systèmes développés en parallèle lors de trois thèses : SAVER (Gatto, Gleizes et Elicegui 2013) pour l'optimisation énergétique de bâtiments, AMADEUS (Guivarch, Camps et Péninou 2012) pour le contrôle de systèmes ambiants, et donc ESCHER pour le contrôle de systèmes complexes. Ces quatre AMAS présentent une architecture (en termes de types d'agents et de la fonction que chacun remplit) similaire, ils diffèrent majoritairement par ce qui pourrait être considéré comme des détails :

- Obsidian peut être vu comme une version alpha des trois autres systèmes. Il ne possède pas d'Agents Critères, et bon nombre de mécanismes (comme la consigne dynamique) sont absents. En outre, quelques hypothèses sur l'activité désirée du système diffèrent de celles de ESCHER. Ainsi, les deux systèmes ont très peu de SNC en commun. Par exemple, Obsidian interdit que des plages de validité se recouvrent entre Agents Contextes, et cherche à fusionner ces derniers.
- AMADEUS s'interdit de faire des actions qu'il n'a jamais observées au préalable dans son environnement. Aussi, il n'explore pas l'espace d'états, mais attend qu'un humain lui montre l'action à faire. Il retient le contexte dans lequel cette action lui a été montrée, et la reproduit en anticipant si ce contexte (ou une généralisation de celui-ci) réapparaît. Le but du système est de se substituer aux humains occupant un bâtiment pour la gestion de ce dernier (allumer la lumière, le chauffage, ouvrir les volets, etc). Cette tâche globale fait apparaître des agents (autres que les Agents Contextes) différents de ceux de ESCHER.

– SAVER est le système le plus proche de ESCHER. La principale différence réside dans la définition d'une action : pour ESCHER il s'agit d'une variation sur une entrée, pour SAVER c'est une affectation de valeur. De cette différence découle une utilisation différente des Agents Contextes et des Agents Contrôleurs, et ainsi des SNC différentes.

Avoir plusieurs systèmes similaires, réutilisant en partie les mêmes types d'agents est une première dans l'histoire des AMAS et constitue un cas d'étude intéressant. ESCHER est le seul à avoir eu l'opportunité d'être appliqué sur un cas réel, mais les résultats obtenus par ailleurs avec ces systèmes semblent bien différents les uns des autres. Un problème résoluble par ESCHER peut tenir SAVER en échec, et inversement. L'étude de ces divergences peut être très instructive sur les liens entre les éléments du niveau local (les agents et leurs interactions, ainsi que les SNC) et la fonction émergente d'un AMAS. Les premières constatations informelles à ce sujet semblent indiquer que l'émergence réside dans les détails. Par exemple, il semblerait que définir une action comme une variation soit un avantage pour l'apprentissage et la première convergence, mais pas pour le contrôle à proprement parler. SAVER a besoin de beaucoup plus de *feedbacks* et d'itérations que ESCHER pour suivre une consigne en créneau, mais lui est plus fidèle par la suite (aucune oscillation et temps d'établissement plus rapide).

Outre l'étude comparative poussée désormais permise, ces quatre systèmes ouvrent la voie à la véritable réutilisation d'un type d'agent, directement d'un système à l'autre. En effet, les Agents Contextes semblent appelés à être généralisés, standardisés, et à devenir un modèle générique d'agent pour l'apprentissage au sein d'un AMAS. Ils seront très probablement réutilisés dans les prochaines thèses de l'équipe SMAC.

Les SMA, et à plus forte raison les AMAS, sont une technologie très jeune en comparaison de bon nombre de techniques utilisées dans le contrôle intelligent, notamment les réseaux de neurones et les algorithmes génétiques. Il en découle un certain manque de maturité théorique. En effet, même en connaissant parfaitement le système contrôlé, rien ne prouve que ESCHER va converger s'il est appliqué. Une perspective à long terme est de faire la preuve de ce type de propriétés. Une première étape consiste en la formalisation du système, ébauchée dans le chapitre 4. S'il semble plus facile à résoudre sur un système particulier, ce problème concerne les AMAS de manière générale. Aussi, c'est un travail au long cours qui n'est pas abordé de manière spécifique à ESCHER (Graja et al. 2014).

Et Gödel dans tout ça ?

Le lecteur qui aura deviné d'où vient l'inspiration du nom des systèmes présentés dans cette thèse [1] peut légitimement se poser la question. Le théorème de l'incomplétude de Gödel stipule qu'une théorie mathématique suffisamment puissante pour formaliser l'arithmétique ne peut être à la fois complète (capable d'exprimer toutes les assertions possibles) et consistante (qui ne contient pas d'incohérence). Il est hasardeux d'étendre ce théorème hors du cadre mathématique strict auquel il s'applique, mais je vais tout de même m'y risquer afin d'illustrer mon propos. Une des perspectives à très long terme des

1. Du livre *Gödel, Escher, Bach : The Eternal Golden Braid* (Hofstadter 1979)

AMAS est d'obtenir un système capable de tout apprendre, c'est-à-dire de faire émerger sa propre fonction. Cet AMAS idéal serait capable d'effectuer et de s'adapter à toute tâche que l'on pourrait attendre de n'importe quel système informatique. Lorsque l'on construit un programme, la pratique habituelle est de s'assurer de la consistance de ce dernier. Un tel AMAS (s'il existe) prendrait l'envers de ce constat. Ce serait un système qui résoudrait des SNC en permanence. Il maintiendrait un équilibre dynamique pour exercer une fonction donnée, et basculerait d'un équilibre à l'autre en fonction de son environnement. Il ne serait jamais réellement consistant, mais virtuellement complet, capable de faire face à n'importe quel problème, de s'adapter à tout (et d'obtenir le meilleur score au test d'intelligence universelle, Legg et Hutter 2007).

Un des espoirs à très long terme de la théorie des AMAS est d'échapper à Gödel en proposant une théorie purement locale, et localement trop simple pour que le théorème puisse s'appliquer. En l'état, la théorie permet déjà de repousser les limites de la complexité des logiciels concevables grâce à sa focalisation sur le niveau local, plus simple, des agents, et en laissant à ceux-ci le contrôle de leur processus d'organisation. Ainsi, la complexité des systèmes produits émerge maintenant des agents, et non plus de l'esprit du concepteur. Le défi que relève l'approche AMAS est de maîtriser l'auto-organisation, de trouver quels mécanismes locaux vont faire émerger les phénomènes globaux attendus. La coopération est un très bon moyen d'y parvenir, car c'est un concept intuitif, qui fournit des lignes de conduites relativement précises. Cependant, cet avantage a un coût. Les concepts intuitifs renferment des ambiguïtés parfois bien cachées sous une couche de "sens commun", ce qui les rend très difficiles à formaliser. Or, une formalisation poussée de la théorie AMAS semble indispensable, ne serait-ce que pour montrer la convergence sur certains types de problèmes. Peut-être la coopération est-elle une notion trop antropomorphique ? Du point de vue d'un concepteur de SMA, la véritable question est plutôt de savoir si la maîtrise d'un phénomène émergent, par des agents purement locaux qui n'en ont pas la connaissance, est possible en toute circonstance. Voilà le pari qui fait tout le sel de l'approche AMAS !

Si cette thèse a apporté quelques réponses à la problématique de l'apprentissage dans le cadre du contrôle de systèmes complexes, elle a également mis en évidence quelques problèmes. Et ce n'est pas un mal. Une réponse est par nature statique, inerte. Ce sont les questions, les problèmes, qui provoquent (et suivent) le mouvement. Une question prend de la valeur avec le temps, une réponse en perd [2]. C'est pourquoi il est toujours préférable de poser une question que d'apporter une réponse. N'est-ce pas ?

2. Cette idée est très joliment développée par l'artiste Kostas Kiriakakis, à cette adresse : http ://kiriakakis.net/comics/mused/a-day-at-the-park

Références bibliographiques

Aamodt, Agnar et Plaza, Enric (1994). « Case-based reasoning : Foundational issues, methodological variations, and system approaches ». In : *AI communications* 7.1, p. 39–59 (cf. p. 45).

Abraham, Ajith, Grosan, Crina et Ramos, Vitorino (2006). *Stigmergic optimization*. T. 31. Springer (cf. p. 79).

Ackley, David H., Hinton, Geoffrey E. et Sejnowski, Terrence J. (1985). « A learning algorithm for Boltzmann machines ». In : *Cognitive science* 9.1, p. 147–169 (cf. p. 41).

Alahakoon, Damminda, Halgamuge, Saman K. et Srinivasan, Bala (2000). « Dynamic self-organizing maps with controlled growth for knowledge discovery ». In : *IEEE Transactions on Neural Networks* 11.3, p. 601–614 (cf. p. 39).

Aldewereld, Huib et Dignum, Virginia (2011). « OperettA : Organization-oriented development environment ». In : *Languages, Methodologies, and Development Tools for Multi-Agent Systems*. Springer, p. 1–18 (cf. p. 75).

Altekar, Gautam, Dwarkadas, Sandhya, Huelsenbeck, John P. et Ronquist, Fredrik (2004). « Parallel metropolis coupled Markov chain Monte Carlo for Bayesian phylogenetic inference ». In : *Bioinformatics* 20.3, p. 407–415 (cf. p. 33).

Andre, David et Teller, Astro (1999). « Evolving team darwin united ». In : *RoboCup-98 : Robot soccer world cup II*. Springer, p. 346–351 (cf. p. 73).

Arimoto, Suguru, Kawamura, Sadao et Miyazaki, Fumio (1984). « Bettering operation of dynamic systems by learning : A new control theory for servomechanism or mechatronics systems ». In : *The 23rd IEEE Conference on Decision and Control.* T. 23. IEEE, p. 1064–1069 (cf. p. 19).

Arthur, David et Vassilvitskii, Sergei (2007). « k-means++ : The advantages of careful seeding ». In : *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial et Applied Mathematics, p. 1027–1035 (cf. p. 37).

Ashby, William Ross (1956). *An Introduction to Cybernetics*. London, UK : Chapman & Hall (cf. p. 6, 87).

Astrid, Patricia, Huisman, Leo, Weiland, Siep et Backx, Ton A.C.P.M. (2002). « Reduction and predictive control design for a computational fluid dynamics model ». In : *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*. T. 3. IEEE, p. 3378–3383 (cf. p. 52).

Astrom, Karl Johan et Hagglund, Tore (1995). *PID Controllers : Theory, Design, and Tuning*. Second. Research Triangle Park, NC : Instrument Society of America (cf. p. 12).

Atkinson, Chris et Mott, Gregory (2005). « Dynamic model-based calibration optimization : An introduction and application to diesel engines ». In : *SAE 2005 World Congress & Exhibition Technical Papers* (cf. p. 61).

Auer, Peter, Cesa-Bianchi, Nicolò et Fischer, Paul (2002). « Finite-time analysis of the multiarmed bandit problem ». In : *Machine learning* 47.2-3, p. 235–256 (cf. p. 47).

Auger, Anne et Hansen, Nikolaus (2005). « A restart CMA evolution strategy with increasing population size ». In : *Evolutionary Computation, 2005. The 2005 IEEE Congress on*. T. 2. IEEE, p. 1769–1776 (cf. p. 28).

Ayadi, Badii et Benhadj, BJ (2005). « MIMO PID controllers synthesis using orthogonal functions ». In : *Preprints of the 16th IFAC World Congress, Prague, Czech Republic*. (Cf. p. 14).

Báez, José, Stratulat, Tiberiu et Ferber, Jacques (2005). « Un modèle institutionnel pour SMA organisationnel ». In : *Systèmes Multi-Agents : Vers la Conception de Systèmes Artificiels Socio-Mimétiques (JFSMA'05)* (cf. p. 75).

Barreto, Guilherme A. et Araujo, Aluizio F. R. (2004). « Identification and control of dynamical systems using the self-organizing map ». In : *Neural Networks, IEEE Transactions on* 15.5, p. 1244–1259 (cf. p. 52).

Begum, Shahina, Ahmed, Mobyen Uddin, Funk, Peter, Xiong, Ning et Folke, Mia (2011). « Case-based reasoning systems in the health sciences : a survey of recent trends and developments ». In : *Systems, Man, and Cybernetics, Part C : Applications and Reviews, IEEE Transactions on* 41.4, p. 421–434 (cf. p. 47).

Ben-Gal, Irad (2007). « Bayesian networks ». In : *Encyclopedia of statistics in quality and reliability* (cf. p. 32).

Bennett, Kristin et Demiriz, Ayhan (1999). « Semi-supervised support vector machines ». In : *Advances in Neural Information processing systems*, p. 368–374 (cf. p. 36).

Blum, Avrim et Mitchell, Tom (1998). « Combining labeled and unlabeled data with co-training ». In : *Proceedings of the eleventh annual conference on Computational learning theory*. ACM, p. 92–100 (cf. p. 36).

Bongard, Josh C. (2000). « The legion system : A novel approach to evolving heterogeneity for collective problem solving ». In : *Genetic Programming*. Springer, p. 16–28 (cf. p. 73).

Bonjean, Noelie, Mefteh, Wafa, Gleizes, Marie-Pierre, Maurel, Christine et Migeon, Frédéric (2013). *ADELFE 2.0*. français. Rapport de recherche IRIT/RR–2013-18–FR. Université Paul Sabatier, Toulouse : IRIT, jan. 2013 (cf. p. 85).

Borman, G. (1964). « Mathematical simulation of internal combustion engine processes and performance including comparison with experiment ». Thèse de doct. Univ. of Wisconsin (cf. p. 59).

Bourjot, Christine, Desor, Didier et Chevrier, Vincent (2011). « Stigmergy ». In : *Self-organising Software*. Sous la dir. de Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes et Anthony Karageogos. Natural Computing Series. Springer, p. 123–138 (cf. p. 80).

Braziunas, Vytautas (1992). *Design of the MRAC Scheme with PID Controllers for Robotic Manipulators*. Northern Illinois University (cf. p. 16).

Breiman, Leo (1996). « Bagging predictors ». In : *Machine learning* 24.2, p. 123–140 (cf. p. 34).

Breiman, Leo (2001). « Random forests ». In : *Machine learning* 45.1, p. 5–32 (cf. p. 35).

Breiman, Leo, Friedman, Jerome H., Olshen, Richard A. et Stone, Charles J. (1984). « Classification and Regression Trees ». In : *Wadsworth International Group* (cf. p. 24).

Buche, Cédric, Septseault, Cyril et De Loor, Pierre (2006). « Les systèmes de classeurs : Une présentation générale ». In : *TSI. Technique et science informatiques* 25.8-9, p. 963–990 (cf. p. 44).

Bull, Larry et O'Hara, Toby (2002). « Accuracy-based Neuro And Neuro-fuzzy Classifier Systems. » In : *GECCO*. T. 2, p. 905–911 (cf. p. 44).

Bull, Larry, Sha'Aban, Jeanan, Tomlinson, Andy, Addison, Jerveada Dixon et Heydecker, Benjamin G. (2004). « Towards distributed adaptive control for road traffic junction signals using learning classifier systems ». In : *Applications of Learning Classifier Systems*. Springer, p. 276–299 (cf. p. 55).

Camazine, Scott, Deneubourg, Jean-Louis, Franks, Nigel R., Sneyd, James, Theraulaz, Guy et Bonabeau, Eric (2003). *Self-Organization in Biological Systems.* Princeton University Pres (cf. p. 78).

Carvajal, James, Chen, Guanrong et Ogmen, Haluk (2000). « Fuzzy PID controller : Design, performance evaluation, and stability analysis ». In : *Information Sciences* 123.3–4, p. 249–270 (cf. p. 14).

Casals, Alicia (2013). « Adaptive Control in Neurorehabilitation ». In : *Converging Clinical and Engineering Research on Neurorehabilitation*. Springer, p. 123–127 (cf. p. 20).

Chang, Wei-Der, Hwang, Rey-Chue et Hsieh, Jer-Guang (2002). « A self-tuning PID control for a class of nonlinear systems based on the Lyapunov approach ». In : *Journal of Process Control* 12.2, p. 233–242 (cf. p. 14).

Chen, Sheng, Billings, Stephen A. et Luo, Wan (1989). « Orthogonal least squares methods and their application to non-linear system identification ». In : *International Journal of control* 50.5, p. 1873–1896 (cf. p. 16).

Choy, Min Chee, Srinivasan, Dipti et Cheu, Ruey Long (2006). « Neural networks for continuous online learning and control ». In : *Neural Networks, IEEE Transactions on* 17.6, p. 1511–1531 (cf. p. 55).

Comon, Pierre et Jutten, Christian (2010). *Handbook of Blind Source Separation : Independent component analysis and applications*. Elsevier (cf. p. 38).

Coomans, Danny et Massart, Désiré Luc (1982). « Alternative k-nearest neighbour rules in supervised pattern recognition : Part 1. k-Nearest neighbour classification by using alternative voting rules ». In : *Analytica Chimica Acta* 136, p. 15–27 (cf. p. 23).

Cornuéjols, Antoine et Miclet, Laurent (2010). *Apprentissage artificiel : Concepts et algorithmes, 2ème édition*. Eyrolles (cf. p. 21, 42).

Cover, Thomas M. (1991). « Universal portfolios ». In : *Mathematical finance* 1.1, p. 1–29 (cf. p. 24).

Dabo, Marcelin, Langlois, Nicolas, Respondek, Witold et Chafouk, Houcine (2008). « NCGPC with dynamic extension applied to a Turbocharged Diesel Engine ». In : *Proceedings of the International Federation of Automatic Control 17th World Congress*, p. 12065–12070 (cf. p. 60).

Dahl, George, Ranzato, Marc'Aurelio, Mohamed, Abdel-rahman et Hinton, Geoffrey E. (2010). « Phone recognition with the mean-covariance restricted Boltzmann machine ». In : *Advances in neural information processing systems*, p. 469–477 (cf. p. 41).

De Wolf, Tom et Holvoet, Tom (2005). « Emergence versus self-organisation : Different concepts but promising when combined ». In : *Engineering self-organising systems*. Springer, p. 1–15 (cf. p. 77).

Del Re, Luigi, Allgöwer, Frank, Glielmo, Luigi, Guardiola, Carlos et Kolmanovsky, Ilya (2010). *Automotive model predictive control : models, methods and applications*. T. 402. Springer (cf. p. 58).

Demazeau, Yves (1995). « From interactions to collective behaviour in agent-based systems ». In : *Proceedings of the 1st European Conference on Cognitive Science*. Citeseer (cf. p. 70).

Deng, Jiamei, Becerra, Victor M. et Stobart, Richard (2009). « Input constraints handling in an MPC/feedback linearization scheme ». In : *International Journal of Applied Mathematics and Computer Science* 19.2, p. 219–232 (cf. p. 18).

Di Marzo Serugendo, Giovanna, Gleizes, Marie-Pierre et Karageorgos, Anthony, éds. (2011). *Self-organising Software - From Natural to Artificial Adaptation*. Natural Computing Series. Springer, oct. 2011 (cf. p. 78).

Domingos, Pedro et Pazzani, Michael (1997). « On the optimality of the simple Bayesian classifier under zero-one loss ». In : *Machine learning* 29.2-3, p. 103–130 (cf. p. 35).

Dondio, Pierpaolo, Barrett, Stephen, Weber, Stefan et Seigneur, Jean Marc (2006). « Extracting trust from domain analysis : A case study on the wikipedia project ». In : *Autonomic and Trusted Computing*. Springer, p. 362–373 (cf. p. 81).

Dorigo, Marco, Di Caro, Gianni et Gambardella, Luca M. (1999). « Ant algorithms for discrete optimization ». In : *Artificial life* 5.2, p. 137–172 (cf. p. 80).

Elidan, Gal, Nachman, Iftach et Friedman, Nir (2007). « "Ideal Parent" structure learning for continuous variable Bayesian networks ». In : *Journal of Machine Learning Research* 8, p. 1799–1833 (cf. p. 33).

Elkan, Charles (2003). « Using the Triangle Inequality to Accelerate k-Means ». In : *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, p. 147–153 (cf. p. 37).

Fabri, Simon G. et Bugeja, Marvin K. (2013). « Kalman Filter-based Estimators for Dual Adaptive Neural Control : A Comparative Analysis of Execution Time and Performance Issues ». In : *Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2013)*. Reykjavik, Iceland, juil. 2013, p. 169–176 (cf. p. 53).

Fan, Ling Ling et Song, Yong Duan (2012). « Neuro-adaptive model-reference fault-tolerant control with application to wind turbines ». In : *Control Theory & Applications, IET* 6.4, p. 475–486 (cf. p. 51).

Feldbaum, Alexander Aronovich (1961). « Dual Control Theory, I-IV ». In : *Automation Remote Control* 21-22 (cf. p. 18).

Ferber, Jacques (1999). *Multi-agent systems : an introduction to distributed artificial intelligence*. Addison-Wesley Reading (cf. p. 65).

Ferber, Jacques, Gutknecht, Olivier et Michel, Fabien (2004). « From agents to organizations : an organizational view of multi-agent systems ». In : *Agent-Oriented Software Engineering IV*. Springer, p. 214–230 (cf. p. 75).

Ferreau, Hans Joachim (2006). « An Online Active Set Strategy for Fast Solution of Parametric Quadratic Programs with Applications to Predictive Engine Control ». Thèse de doct. Université de Heidelberg (cf. p. 60).

Fiesler, Emile (1996). « Neural Network Topologies" ». In : *The Handbook of Neural Computation*. Sous la dir. d'E. Fiesler et R. Beale (cf. p. 29).

Filipović, Vladimir (2012). « Fine-grained tournament selection operator in genetic algorithms ». In : *Computing and Informatics* 22.2, p. 143–161 (cf. p. 27).

Finin, Tim, Fritzson, Richard, McKay, Don et McEntire, Robin (1994). « KQML as an agent communication language ». In : *Proceedings of the third international conference on Information and knowledge management*. ACM, p. 456–463 (cf. p. 70).

Floreano, Dario, Dürr, Peter et Mattiussi, Claudio (2008). « Neuroevolution : from architectures to learning ». In : *Evolutionary Intelligence* 1.1, p. 47–62 (cf. p. 32).

Freund, Yoav (2001). « An adaptive version of the boost by majority algorithm ». In : *Machine learning* 43.3, p. 293–318 (cf. p. 35).

Freund, Yoav et Schapire, Robert E. (1997). « A decision-theoretic generalization of on-line learning and an application to boosting ». In : *Journal of computer and system sciences* 55.1, p. 119–139 (cf. p. 35).

Gao, Ying, Kipling, Katie, Glassey, Jarka, Willis, Mark, Montague, Gary, Zhou, Yuhong et Titchener-Hooker, Nigel J. (2010). « Application of agent-based system for bioprocess description and process improvement ». In : *Biotechnology progress* 26.3, p. 706–716 (cf. p. 72).

Gatto, François, Gleizes, Marie-Pierre et Elicegui, Lucas (2013). « Saver : Self-adaptive Energy Saver ». In : *European Workshop on Multi-Agent Systems (EUMAS), Toulouse, France* (cf. p. 174).

Georgé, Jean-Pierre, Gleizes, Marie-Pierre et Camps, Valérie (2011). « Cooperation ». In : *Self-organising Software*. Sous la dir. de Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes et Anthony Karageogos. Natural Computing Series. Springer Berlin Heidelberg, p. 7–32 (cf. p. 82, 84, 87).

Gerber, Christian, Siekmann, Jörg et Vierke, Gero (1999). *Holonic multi-agent systems*. eng. Rapp. tech. Postfach 151141, 66041 Saarbrücken : Universitäts- und Landesbibliothek (cf. p. 80).

Gleizes, Marie-Pierre, Camps, Valérie, Karageorgos, Anthony et Di Marzo Serugendo, Giovanna (2011). « Agents and Multi-Agent Systems ». In : *Self-organising Software*. Springer, p. 105–119 (cf. p. 66).

Glize, Pierre (2001). « L'Adaptation des Systèmes à Fonctionnalité Émergente par Auto-Organisation Coopérative ». Habilitation à diriger des recherches. Toulouse, France : Université Paul Sabatier, juin 2001 (cf. p. 83).

Glodek, Michael, Tschechne, Stephan, Layher, Georg, Schels, Martin, Brosch, Tobias, Scherer, Stefan, Kächele, Markus, Schmidt, Miriam, Neumann, Heiko, Palm, Günther et Schwenker, Friedhelm (2011). « Multiple classifier systems for the classification of audio-visual emotional states ». In : *Affective Computing and Intelligent Interaction*. Springer, p. 359–368 (cf. p. 45).

Goldstein, Jeffrey (1999). « Emergence as a construct : History and issues ». In : *Emergence* 1.1, p. 49–72 (cf. p. 77).

Graja, Zeineb, Migeon, Frédéric, Maurel, Christine, Gleizes, Marie-Pierre, Laibinis, Linas, Regayeg, Amira et Kacem, Ahmed Hadj (2014). « A Pattern based Modelling for Self-Organizing Multi-Agent Systems with Event-B ». In : *International Conference on Agents and Artificial Intelligence (ICAART), Angers, France* (cf. p. 175).

Griffiths, Thomas L. et Yuille, Alan (2008). « A primer on probabilistic inference ». In : *The probabilistic mind : Prospects for Bayesian cognitive science*, p. 33–57 (cf. p. 32).

Guivarch, Valérian, Camps, Valérie et Péninou, André (2012). « Context awareness and adaptation in ambient systems by an adaptive multi-agent approach ». In : *International Joint Conference on Ambient Intelligence* (cf. p. 174).

Guo, Gongde, Wang, Hui, Bell, David, Bi, Yaxin et Greer, Kieran (2003). « KNN Model-Based Approach in Classification ». In : *On The Move to Meaningful Internet Systems 2003 : CoopIS, DOA, and ODBASE*. Sous la dir. de Robert Meersman, Zahir Tari et Douglas C. Schmidt. T. 2888. Lecture Notes in Computer Science. Springer Berlin Heidelberg, p. 986–996 (cf. p. 23).

Hartigan, John A. et Wong, Manchek A. (1979). « Algorithm AS 136 : A k-means clustering algorithm ». In : *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1, p. 100–108 (cf. p. 36).

Hastie, Trevor, Tibshirani, Robert, Friedman, Jerome et Franklin, James (2005). « The elements of statistical learning : data mining, inference and prediction ». In : *The Mathematical Intelligencer* 27.2, p. 83–85 (cf. p. 35).

Heckerman, David (2008). *A tutorial on learning with Bayesian networks*. Springer (cf. p. 33).

Heinen, Milton Roberto et Engel, Paulo Martins (2010). « An incremental probabilistic neural network for regression and reinforcement learning tasks ». In : *Artificial Neural Networks–ICANN 2010*. Springer, p. 170–179 (cf. p. 47).

Heragu, Sunderesh S., Graves, Robert J., Kim, Byung-In et St Onge, Art (2002). « Intelligent agent based framework for manufacturing systems control ». In : *Systems, Man and Cybernetics, Part A : Systems and Humans, IEEE Transactions on* 32.5, p. 560–573 (cf. p. 72, 80).

Hertz, John A., Krogh, Anders S. et Palmer, Richard G. (1991). *Introduction to the Theory of Neural Computation*. T. 1. Basic Books (cf. p. 40).

Heylighen, Francis (2001). « The science of self-organization and adaptivity ». In : *The Encyclopedia of Life Support Systems* 5.3, p. 253–280 (cf. p. 77).

Hinton, Geoffrey E. (2010). « A practical guide to training restricted Boltzmann machines ». In : *Momentum* 9.1 (cf. p. 41).

Hinton, Geoffrey E., Osindero, Simon et Teh, Yee-Whye (2006). « A fast learning algorithm for deep belief nets ». In : *Neural computation* 18.7, p. 1527–1554 (cf. p. 41).

Hoang, Thi-Thanh-Ha, Occello, Michel et Jamont, Jean-Paul (2011). « A Generic Recursive Multiagent Model to Simplify Large Scale Multi-level Systems Observation ». In : *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Volume 02*. WI-IAT '11. Washington, DC, USA : IEEE Computer Society, p. 155–158 (cf. p. 71).

Hofstadter, Douglas Richard (1979). *Gödel, Escher, Bach : An Eternal Golden Braid*. Basic Books, Inc. (cf. p. 175).

Holland, John H. (1975). *Adaptation in Natural and Artificial Systems : An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. University of Michigan Press. Reprinted in 1992 by MIT Press (cf. p. 26).

Hopfield, John J. (1982). « Neural networks and physical systems with emergent collective computational abilities ». In : *Proceedings of the national academy of sciences* 79.8, p. 2554–2558 (cf. p. 39).

Ishida, Yoshiteru (2004). *Immunity-based systems : a design perspective*. Springer (cf. p. 81).

Isidori, Alberto (1999). *Nonlinear control systems*. T. 2. Springer (cf. p. 18).

Jaeger, Herbert (2001). « The" echo state" approach to analysing and training recurrent neural networks-with an erratum note' ». In : *German National Research Center for Information Technology GMD Technical Report* 148 (cf. p. 31).

Jankovic, M. et Kolmanovsky, I. (2000). « Constructive Lyapunov control design for turbocharged diesel engines ». In : *IEEE Transactions on Control Systems Technology* 8.2, p. 288–299 (cf. p. 59).

Jelasity, Márk, Voulgaris, Spyros, Guerraoui, Rachid, Kermarrec, Anne-Marie et Van Steen, Maarten (2007). « Gossip-based peer sampling ». In : *ACM Transactions on Computer Systems (TOCS)* 25.3, p. 8 (cf. p. 81).

Jesus, Isabel S. et Barbosa, Ramiro S. (2013). « Tuning of Fuzzy Fractional PD^{β}+I Controllers by Genetic Algorithm ». In : *Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2013)*. Reykjavik, Iceland, juil. 2013, p. 282–287 (cf. p. 50).

Jin, Yaochu (2005). « A comprehensive survey of fitness approximation in evolutionary computation ». In : *Soft computing* 9.1, p. 3–12 (cf. p. 27).

Juan, Alfons et Vidal, Enrique (2000). « Comparison of four initialization techniques for the k-medians clustering algorithm ». In : *Advances in Pattern Recognition*. Springer, p. 842–852 (cf. p. 37).

Kalenka, Susanne et Jennings, Nicholas R. (1999). « Socially responsible decision making by autonomous agents ». In : *Cognition, Agency and Rationality*. Springer, p. 135–149 (cf. p. 82).

Kalman, Rudolph E. et Bucy, Richard S. (1961). « New results in linear filtering and prediction theory ». In : *Journal of Basic Engineering* 83.3, p. 95–108 (cf. p. 54).

Kanungo, Tapas, Mount, David M., Netanyahu, Nathan S., Piatko, Christine D., Silverman, Ruth et Wu, Angela Y. (2002). « An efficient k-means clustering algorithm : Analysis and

implementation ». In : *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24.7, p. 881–892 (cf. p. 37).

Kauffman, Stuart A. et Smith, Robert G. (1986). « Adaptive automata based on Darwinian selection ». In : *Physica D : Nonlinear Phenomena* 22.1, p. 68–82 (cf. p. 28).

Kaynak, Cenk et Alpaydin, Ethem (2000). « Multistage cascading of multiple classifiers : One man's noise is another man's data ». In : *Proceedings of the 17th International Conference on Machine Learning*. Citeseer, p. 455–462 (cf. p. 35).

Koestler, Arthur (1967). *The ghost in the machine.* Penguin Group (cf. p. 80).

Kohonen, Teuvo (2001). *Self-Organizing Maps, Third edition*. Information sciences. Berlin : Springer-Verlag (cf. p. 38).

Konstantinov, Konstantin B., Aarts, Robert et Yoshida, Toshiomi (1993). « Expert systems in bioprocess control : requisite features ». In : *Bioprocess design and control*. Springer, p. 169–191 (cf. p. 48).

Kramer, Oliver (2010). « Evolutionary self-adaptation : a survey of operators and strategy parameters ». In : *Evolutionary Intelligence* 3.2, p. 51–65 (cf. p. 28).

Kruchten, Philippe (2004). *The rational unified process : an introduction*. Addison-Wesley Professional (cf. p. 85).

Kuhn, Harold. W. et Tucker, Albert. W. (1951). « Nonlinear programming ». In : *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, 1950*. Berkeley et Los Angeles : University of California Press, p. 481–492 (cf. p. 25).

Lauritzen, Steffen L. et Spiegelhalter, David J. (1988). « Local computations with probabilities on graphical structures and their application to expert systems ». In : *Journal of the Royal Statistical Society. Series B (Methodological)*, p. 157–224 (cf. p. 33).

Lazkano, Elena, Sierra, Basilio, Astigarraga, Aitzol et Martinez-Otzeta, José Maria (2007). « On the use of Bayesian Networks to develop behaviours for mobile robots ». In : *Robotics and Autonomous Systems* 55.3, p. 253–265 (cf. p. 57).

Lee, Yongho, Park, Sunwon et Lee, Moonyong (1998). « PID Controller Tuning To Obtain Desired Closed Loop Responses for Cascade Control Systems ». In : *Industrial & Engineering Chemistry Research* 37.5, p. 1859–1865 (cf. p. 14).

Legg, Shane et Hutter, Marcus (2007). « Universal intelligence : A Definition of Machine Intelligence ». In : *Minds and Machines* 17.4, p. 391–444 (cf. p. 176).

Leitão, Paulo (2009). « Agent-based distributed manufacturing control : A state-of-the-art survey ». In : *Engineering Applications of Artificial Intelligence* 22.7, p. 979–991 (cf. p. 72).

Leitão, Paulo et Restivo, Francisco (2006). « ADACOR : A holonic architecture for agile and adaptive manufacturing control ». In : *Computers in industry* 57.2, p. 121–130 (cf. p. 80).

Lemouzy, Sylvain, Camps, Valérie. et Glize, Pierre (2011). « Principles and Properties of a MAS Learning Algorithm : A Comparison with Standard Learning Algorithms Applied to Implicit Feedback Assessment ». In : *2011 International Conference on Web Intelligence and Intelligent Agent Technology*. T. 2. Août 2011, p. 228–235 (cf. p. 108).

Liao, Shu-Hsien (2005). « Expert system methodologies and applications—a decade review from 1995 to 2004 ». In : *Expert systems with applications* 28.1, p. 93–103 (cf. p. 48).

Lomax, Susan et Vadera, Sunil (2013). « A survey of cost-sensitive decision tree induction algorithms ». In : *ACM Comput. Surv.* 45.2 (mar. 2013), 16 :1–16 :35 (cf. p. 24).

Lu, Haiping, Plataniotis, Konstantinos N. et Venetsanopoulos, Anastasios N. (2011). « A survey of multilinear subspace learning for tensor data ». In : *Pattern Recognition* 44.7, p. 1540–1551 (cf. p. 38).

Lukoševičius, Mantas et Jaeger, Herbert (2009). « Survey : Reservoir computing approaches to recurrent neural network training ». In : *Comput. Sci. Rev.* 3.3 (août 2009), p. 127–149 (cf. p. 31).

Maass, Wolfgang, Natschläger, Thomas et Markram, Henry (2002). « Real-time computing without stable states : A new framework for neural computation based on perturbations ». In : *Neural computation* 14.11, p. 2531–2560 (cf. p. 31).

Malikopoulos, Andreas A, Assanis, Dennis N et Papalambros, Panos Y (2009). « Real-time self-learning optimization of diesel engine calibration ». In : *Journal of Engineering for Gas Turbines & Power* 131.2, p. 22803 (cf. p. 61, 87).

Mareels, Iven M.Y., Anderson, Brian D.O., Bitmead, Robert R., Bodson, Marc et Sastry, Shankar S. (1986). « Revisiting the MIT rule for adaptive control ». In : *Proceedings of the 2nd IFAC Workshop on Adaptive Systems in Control and Signal Processing*, p. 161–166 (cf. p. 16).

McCulloch, Warren S. et Pitts, Walter (1943). « A logical calculus of the ideas immanent in nervous activity ». In : *The Bulletin of Mathematical Biophysics* 5.4, p. 115–133 (cf. p. 29).

McKay, Robert I., Hoai, Nguyen Xuan, Whigham, Peter Alexander, Shan, Yin et O'Neill, Michael (2010). « Grammar-based Genetic Programming : a survey ». In : *Genetic Programming and Evolvable Machines* 11.3-4, p. 365–396 (cf. p. 28).

Mefteh, Wafa, Migeon, Frédéric, Gleizes, Marie-Pierre et Gargouri, Faiez (2013). « Simulation Based Design for Adaptive Multi-agent Systems : Extension to the ADELFE Methdology ». In : *Workshops on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE)*, p. 36–38 (cf. p. 174).

Minorsky, Nicolas (1922). « Directional stability of automatically steered bodies ». In : *Journal of ASNE* 34.2, p. 280–309 (cf. p. 3).

Mitchell, Tom Michael (2006). *The discipline of machine learning*. Carnegie Mellon University, School of Computer Science, Machine Learning Department (cf. p. 21).

Mizumoto, Ikuro, Ikeda, Daisuke, Hirahata, Tadashi et Iwai, Zenta (2010). « Design of discrete time adaptive PID control systems with parallel feedforward compensator ». In : *Control Engineering Practice* 18.2, p. 168–176 (cf. p. 14).

Montresor, Alberto, Meling, Hein et Babaoğlu, Özalp (2003). « Messor : Load-Balancing through a Swarm of Autonomous Agents ». In : *Agents and Peer-to-Peer Computing*. Sous la dir. de Gianluca Moro et Manolis Koubarakis. T. 2530. Lecture Notes in Computer Science. Springer Berlin Heidelberg, p. 125–137 (cf. p. 80).

Moré, Jorge J. (1978). « The Levenberg-Marquardt algorithm : implementation and theory ». In : *Numerical analysis*. Springer, p. 105–116 (cf. p. 60).

Mudi, Rajani K., Dey, Chanchal et Lee, Tsu-Tian (2008). « An improved auto-tuning scheme for PI controllers ». In : *ISA Transactions* 47.1, p. 45–52 (cf. p. 13).

Müller, Matthias A., Reble, Marcus et Allgöwer, Frank (2011). « A general distributed MPC framework for cooperative control ». In : *Proceedings of the 18th IFAC World Congress*, p. 7987–7992 (cf. p. 18, 87).

Nikolaou, Michael (2001). « Model Predictive Controllers : A Critical Synthesis of Theory and Industrial Needs ». In : *Advances in Chemical Engineering* 26, p. 131–204 (cf. p. 17).

Nikzadfar, Kamyar, Noorpoor, Alireza et Shamekhi, Amir H. (2012). « Design of an optimal idle speed controller for a turbocharged diesel engine using fuzzy logic method ». In : *Journal of mechanical science and technology* 26.8, p. 2325–2336 (cf. p. 61).

Noël, Victor (2012). « Component-based Software Architectures and Multi-Agent Systems : Mutual and Complementary Contributions for Supporting Software Development ». Thèse de doctorat. Toulouse, France : Université de Toulouse, juillet 2012 (cf. p. 68).

Noël, Victor, Arcangeli, Jean-Paul et Gleizes, Marie-Pierre (2012). « Une approche architecturale à base de composants pour l'implémentation des Systèmes Multi-Agents ». In : *Revue des Nouvelles Technologies de l'Information, Avancées récentes dans le domaines des Architectures Logicielles : articles sélectionnés et étendus de CAL 2011* RNTI-L-6, p. 1–26 (cf. p. 86, 112).

Nørgaard, Magnus, Ravn, Ole, Poulsen, Niels Kjølstad et Hansen, Lars Kai (2000). *Neural Networks for Modelling and Control of Dynamic Systems-A Practitioner's Handbook*. Springer (cf. p. 52).

Noura, Hassan, Theillol, Didier, Ponsart, Jean-Christophe et Chamseddine, Abbas (2009). *Fault-tolerant control systems : Design and practical applications*. Springer (cf. p. 51).

Odell, James (2002). « Agents and complex systems ». In : *Journal of Object Technology* 1.2, p. 35–45 (cf. p. 77).

Odell, James J., Parunak, H. Van Dyke, Fleischer, Mitch et Brueckner, Sven (2003). « Modeling agents and their environment ». In : *Agent-oriented software engineering III*. Springer, p. 16–31 (cf. p. 71).

Oliver, Jonathan J. (1993). « Decision Graphs : An Extension of Decision Trees ». In : *Preliminary papers of the fourth International Workshop on Artificial Intelligence and Statistics : January 3-6, 1993, Ft. Lauderdale, Florida*. Society for Artificial Intelligence et Statistics, p. 343 (cf. p. 24).

Ouladsine, Mustapha, Bloch, Gérard et Dovifaaz, Xavier (2005). « Neural modelling and control of a Diesel engine with pollution constraints ». In : *Journal of Intelligent and Robotic Systems* 41.2-3, p. 157–171 (cf. p. 61).

Owens, David H. et Daley, Steve (2008). « Iterative Learning Control—Monotonicity and Optimization ». In : *International Journal of Applied Mathematics and Computer Science* 18.3, p. 279–293 (cf. p. 20).

Panait, Liviu et Luke, Sean (2005). « Cooperative multi-agent learning : The state of the art ». In : *Autonomous Agents and Multi-Agent Systems* 11.3, p. 387–434 (cf. p. 72).

Pargfrieder, Joachim et Jörgl, H.P. (2002). « An integrated control system for optimizing the energy consumption and user comfort in buildings ». In : *Computer Aided Control System Design, 2002. Proceedings. 2002 IEEE International Symposium on*. IEEE, p. 127–132 (cf. p. 20).

Park, Hae-Sang et Jun, Chi-Hyuck (2009). « A simple and fast algorithm for K-medoids clustering ». In : *Expert Systems with Applications* 36.2, p. 3336–3341 (cf. p. 37).

Patel, Nikita et Upadhyay, Saurabh (2012). « Study of Various Decision Tree Pruning Methods with their Empirical Comparison in WEKA ». In : *International Journal of Computer Applications* 60.12 (cf. p. 25).

Pearl, Judea (1985). « Bayesian Networks : A Model of Self-Activated Memory for Evidential Reasoning ». In : *Proceedings of the 7th Conference of the Cognitive Science Society*, p. 329–334 (cf. p. 32).

Pintelon, Rik et Schoukens, Johan (2004). *System identification : a frequency domain approach*. Wiley-IEEE Press (cf. p. 16).

Potter, Mitchell A. et De Jong, Kenneth A. (2000). « Cooperative coevolution : An architecture for evolving coadapted subcomponents ». In : *Evolutionary computation* 8.1, p. 1–29 (cf. p. 28).

Quinlan, J. Ross (1986). « Induction of decision trees ». In : *Machine Learning* 1.1, p. 81–106 (cf. p. 24).

Rao, Anand S. et Georgeff, Michael P. (1995). « BDI Agents : From Theory to Practice. » In : *ICMAS*. T. 95, p. 312–319 (cf. p. 66).

Rodriguez, Sebastian, Hilaire, Vincent, Gaud, Nicolas, Galland, Stephane et Koukam, Abderrafiâa (2011). « Holonic Multi-Agent Systems ». In : *Self-organising Software*. Sous la dir. de Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes et Anthony Karageorgos. Springer, p. 251–279 (cf. p. 80).

Ronan, Colin Alistair (1983). *The Cambridge illustrated history of the world's science*. Cambridge University Press (cf. p. 2).

Rosenblatt, Frank (1957). *The perceptron, a perceiving and recognizing automaton*. Cornell Aeronautical Laboratory (cf. p. 29).

Rumelhart, David E., Hinton, Geoffrey E. et Williams, Ronald J. (1986). « Learning representations by back-propagating errors ». In : *Nature* 323.6088, p. 533–536 (cf. p. 30).

Russell, Stuart Jonathan et Norvig, Peter (2010). *Artificial intelligence : a modern approach, 3rd edition*. Prentice Hall, Person Education Inc. (cf. p. 22, 70).

Sanchez, Stéphane (2004). « Mécanismes évolutionnistes pour la simulation comportementale d'acteurs virtuels ». Thèse de doct. Université de Toulouse I (cf. p. 45).

Sanza, Cédric (2001). « Evolution d'entités virtuelles coopératives par système de classifieurs ». Thèse de doct. Université de Toulouse III - Paul Sabatier (cf. p. 45).

Sastry, Shankar et Bodson, Marc (1994). *Adaptive control : stability, convergence and robustness*. Courier Dover Publications (cf. p. 16).

Schapire, Robert E. (1990). « The strength of weak learnability ». In : *Machine learning* 5.2, p. 197–227 (cf. p. 35).

Schölkopf, Bernhard et Smola, Alexander J. (2002). *Learning with kernels : support vector machines, regularization, optimization and beyond*. the MIT Press (cf. p. 26).

Shu, Huailin et Pi, Youguo (2000). « PID neural networks for time-delay systems ». In : *Computers and Chemical Engineering* 24.2–7, p. 859–862 (cf. p. 14).

Simonin, Olivier et Gechter, Franck (2006). « An environment-based methodology to design reactive multi-agent systems for problem solving ». In : *Environments for Multi-Agent Systems II*. Springer, p. 32–49 (cf. p. 79).

Soderstrom, Torsten et Stoica, Petre (1988). *System identification*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc. (cf. p. 16).

Sontag, Eduardo D. (1998). *Mathematical control theory : deterministic finite dimensional systems*. Springer (cf. p. 4).

Steck, Harald (2001). « Constraint-based structural learning in Bayesian networks using finite data sets ». Thèse de doct. Technische Universität München, Universitätsbibliothek (cf. p. 34).

Su, Chao-Ton et Shiue, Yeou-Ren (2003). « "Intelligent scheduling controller for shop floor control systems : a hybrid genetic algorithm/decision tree learning approach" ». In : *International Journal of Production Research* 41.12, p. 2619–2641 (cf. p. 57).

Sutton, Richard S. (1996). « Generalization in reinforcement learning : Successful examples using sparse coarse coding ». In : *Advances in neural information processing systems*, p. 1038–1044 (cf. p. 43).

Suykens, Johan A. K., Vandewalle, Joos et De Moor, Bart (2001). « Optimal control by least squares support vector machines ». In : *Neural Networks* 14.1, p. 23–35 (cf. p. 57).

Tannen, Deborah (1999). *The argument culture : Stopping America's war of words*. Ballantine Books New York (cf. p. 84).

Tuyls, Karl, Hoen, Pieter Jan'T et Vanschoenwinkel, Bram (2006). « An evolutionary dynamical analysis of multi-agent learning in iterated games ». In : *Autonomous Agents and Multi-Agent Systems* 12.1, p. 115–153 (cf. p. 73).

Urbanowicz, Ryan J. et Moore, Jason H. (2009). « Learning classifier systems : a complete introduction, review, and roadmap ». In : *Journal of Artificial Evolution and Applications* 2009, p. 1 (cf. p. 44).

Valckenaers, Paul, Saint Germain, Bart, Verstraete, Paul et Van Brussel, Hendrik (2007). « MAS coordination and control based on stigmergy ». In : *Computers in Industry* 58.7, p. 621–629 (cf. p. 80).

Valério, Duarte et Sá da Costa, José (2006). « Tuning of fractional PID controllers with Ziegler–Nichols-type rules ». In : *Signal Processing* 86.10, p. 2771–2784 (cf. p. 13).

Van Der Merwe, Rudolph et Wan, Eric A. (2001). « The square-root unscented Kalman filter for state and parameter-estimation ». In : *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*. T. 6. IEEE, p. 3461–3464 (cf. p. 16).

Vapnik, Vladimir (1995). *The Nature of Statistical Learning Theory*. New York : Springer-Verlag (cf. p. 25).

Videau, Sylvain (2011). « Contrôle de processus dynamiques par systèmes multi-agents adaptatifs : application au contrôle de bioprocédés ». Thèse de doctorat. INSA Toulouse, juillet 2011 (cf. p. 89, 173, 174).

Vilalta, Ricardo et Drissi, Youssef (2002). « A perspective view and survey of meta-learning ». In : *Artificial Intelligence Review* 18.2, p. 77–95 (cf. p. 34).

Visioli, Antonio (2001). « Tuning of PID Controllers With Fuzzy Logic ». In : *IEE Proceedings - Control Theory and Applications* 148.1 (jan. 2001), p. 1–8 (cf. p. 13).

Wang, Chi-Hsu et Hung, Kun-Neng (2010). « Adaptive high-order Hopfield-based neural network tracking controller for uncertain nonlinear dynamical system ». In : *Networking, Sensing and Control (ICNSC), 2010 International Conference on*. IEEE, p. 382–387 (cf. p. 57).

Wang, Yong (2012). « Gauss–Newton method ». In : *Wiley Interdisciplinary Reviews : Computational Statistics* 4.4, p. 415–420 (cf. p. 60).

Wang, Youqing, Gao, Furong et Doyle, Francis J. III (2009). « Survey on iterative learning control, repetitive control, and run-to-run control ». In : *Journal of Process Control* 19.10, p. 1589–1600 (cf. p. 20).

Watkins, Christopher et Dayan, Peter (1992). « Q-learning ». In : *Machine learning* 8.3-4, p. 279–292 (cf. p. 43).

Weyns, Danny, Parunak, H. Van Dyke, Michel, Fabien, Holvoet, Tom et Ferber, Jacques (2005). « Environments for multiagent systems state-of-the-art and research challenges ». In : *Environments for multi-agent systems*. Springer, p. 1–47 (cf. p. 69).

Whitaker, H.Philip, Yamron, Joseph et Kezer, Allen (1958). *Design of Model Reference Adaptive Control Systems for Aircraft*. Issue 164 of Report Massachusetts Institute of Technology Instrumentation Laboratory R. M.I.T. Instrumentation Laboratory (cf. p. 15).

Wiener, Norbet (1948). *Cybernetics or Control and Communication in the Animal and the Machine* (cf. p. 6).

Wilson, Stewart W. (1994). « ZCS : A zeroth level classifier system ». In : *Evolutionary computation* 2.1, p. 1–18 (cf. p. 44).

Wilson, Stewart W. (1995). « Classifier fitness based on accuracy ». In : *Evolutionary computation* 3.2, p. 149–175 (cf. p. 44).

Wittenmark, Björn (2002). « Adaptive dual control ». In : *Control Systems, Robotics and Automation, Encyclopedia of Life Support Systems (EOLSS), Developed under the auspices of the UNESCO* (cf. p. 19).

Wolpert, David H. (1992). « Stacked generalization ». In : *Neural networks* 5.2, p. 241–259 (cf. p. 34).

Wooldridge, Michael et Jennings, Nicholas R. (1995). « Intelligent agents : Theory and practice ». In : *Knowledge engineering review* 10.2, p. 115–152 (cf. p. 65).

Xue, Xiaocen, Dong, Zhanbo, Xiang, Wenguo et Lu, Jianhong (2012). « A new neuro-fuzzy approach for nonlinear system identification based on differential evolution ». In : *9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*. IEEE, p. 409–413 (cf. p. 16).

Zadeh, Lotfi Asker (1965). « Fuzzy sets ». In : *Information and control* 8.3, p. 338–353 (cf. p. 49).

Zadeh, Lotfi Asker (1988). « Fuzzy logic ». In : *Computer* 21.4, p. 83–93 (cf. p. 49).

Zhang, Nevin Lianwen et Poole, David (1996). « Exploiting Causal Independence in Bayesian Network Inference ». In : *Journal of Artificial Intelligence Research* 5, p. 301–328 (cf. p. 33).

Zhu, Ji, Zou, Hui, Rosset, Saharon et Hastie, Trevor (2009). « Multi-class adaboost ». In : *Statistics and its Interface* 2, p. 349–360 (cf. p. 35).

Ziegler, John G. et Nichols, Nathaniel B. (1942). « Optimum settings for automatic controllers ». In : *trans. ASME* 64.11 (cf. p. 13).

Acronymes

ACP *Analyse en Composantes Principales*, techniques d'apprentissage non supervisé, permettant de réduire la dimension de données tout en conservant l'information pertinente.

ADELFE *Atelier de DEveloppement de Logiciels à Fonctionnalité Emergente*, méthode de conception et de développement d'un AMAS.

AMAS *Adaptive Multi-Agent System*, système multi-agent dans lequel les agents fondent leur comportement sur la coopération, permettant la résolution de tâches complexes grâce à l'auto-organisation.

AVT *Adaptive Value Tracker*, outil permettant de suivre une valeur dynamique.

BACH *Builder of Abstract maCHines*, AMAS pour la génération de boîtes noires afin de tester des systèmes d'apprentissage du contrôle, conçu et développé lors de cette thèse.

CO *Monoxyde de carbone*, gaz nocif présent à l'échappement d'un moteur.

ECU *Engine Control Unit*, boîtier électronique embarqué dans un véhicule sur lequel s'exécute un logiciel de contrôle moteur.

EGR *Exhaust Gas Recirculation*, technologie permettant le recyclage comme comburant d'une partie des gaz d'échappement d'un moteur à combustion.

ESCHER *Emergent Self-adaptive Controller for Heat Engine calibRation*, AMAS dédié à l'apprentissage du contrôle de systèmes complexes, conçu, développé, et expérimenté dans le cadre de la calibration d'ECU au cours de cette thèse.

HC *Hydrocarbures*, molécule composée uniquement d'hydrogène et de carbone ; il s'agit dans cette thèse d'un polluant présent dans les gaz d'échappement.

LCS *Learning Classifier System*, système d'apprentissage par renforcement basé sur les algorithmes génétiques.

MAY *Make Agent Yourself*, outil dédié au développement de systèmes multi-agents, permettant la génération de code à partir d'une description d'architecture en composants.

MIAC *Model Identification Adaptive Control*, approche de contrôle de systèmes dans laquelle un modèle du système contrôlé est ajusté au cours du temps.

MIMO *Multiple Input and Multiple Output*, qualifie un système possédant plusieurs entrées et plusieurs sorties.

MPC *Model Predictive Command*, technique de contrôle de systèmes dans laquelle un modèle du système contrôlé est utilisé pour faire des prévisions sur son comportement et explorer les diverses solutions.

MRAC *Model Reference Adaptive Control*, approche de contrôle de systèmes dans laquelle un modèle mathématique sert de référence, et que le système contrôlé doit suivre.

PID *Proportionnel-Intégral-Dérivé*, technique de contrôle se basant sur trois termes liés à l'erreur commise par rapport à la consigne.

SISO *Simple Input and Simple Output*, qualifie un système possédant une seule entrée et une seule sortie.

SMA *Système Multi-Agent*, système composé d'entités autonomes, les agents, qui interagissent au sein d'un environnement qu'elles perçoivent et peuvent modifier.

SNC *Situation de Non-Coopération*, situation dans laquelle un agent d'un AMAS n'est pas capable de remplir sa fonction.

SVM *Support Vector Machines*, ou machines à vecteurs de support, technique d'apprentissage supervisé s'appuyant sur les méthodes d'optimisation mathématiques.

Table des figures

Liste des tableaux

Liste des Algorithmes

ANNEXE A

La fonction barrière

Cette fonction, dont la formulation mathématique a été proposée par des collègues de l'Institut de Mathématiques de Toulouse, est utilisée pour calculer un niveau de criticité en fonction d'une valeur numérique. Le terme "barrière" utilisé ici n'a pas exactement son sens mathématique, mais est imagé et correspond l'utilisation que nous avons de la fonction.

La criticité est une fonction permettant d'évaluer localement par un agent sa difficulté à effectuer les activités qu'il souhaite réaliser (typiquement un objectif à atteindre). Cette valeur de criticité est systématiquement diffusée dans son voisinage à chaque transmission d'information, permettant à chacun d'eux de juger celui qui semble le plus prioritaire à satisfaire (selon l'attitude coopérative). Usuellement la criticité est définie sur $\mathcal{R}$ telle que :

$$f(x) = \begin{cases} \text{criticité maximale si } x \in]-\infty; inf] \\ \text{fonction à trouver si } x \in [inf; sup] \\ \text{criticité maximale si } x \in]sup; +\infty[\end{cases}$$

La fonction à trouver doit être continue et dérivable, rapidement calculable (car utilisée à grande échelle dans un AMAS). Elle doit également avoir peu de paramètres pour son ajustement.

Dans la plage $[inf; sup]$ la criticité est généralement nulle en son centre, et décroissante (respectivement croissante) au voisinage de inf (respectivement sup). La fonction de criticité ci-dessous est définie par morceaux et dépend (outre $critMax$, sa valeur maximale, et inf et sup ses bornes) de deux paramètres : ϵ et η. Il faut $\epsilon > 0$ et $0 < \eta < \epsilon$. En outre, la fonction vérifie :

- $f(inf) = f(sup) = critMax$ et leurs dérivées sont nulles ;
- $f(inf + \epsilon) = f(sup - \epsilon) = 0$ et leurs dérivées sont nulles.

Pour simplifier l'écriture, nous transposons par un changement de variable les bornes dans $[0; sup - inf]$. Et par abus de langage, nous continuons d'appeler sup la borne supérieure du

nouvel intervalle. La fonction barrière de criticité est définie par :

$$f(x) = \begin{cases} critMax & \text{si } x \leq 0 \\ \gamma\frac{(x-\eta)^2}{2\eta} + \gamma(x-\eta) + \delta & \text{si } 0 < x \leq \eta \\ -\gamma\frac{(x-\eta)^2}{2(\epsilon-\eta)} + \gamma(x-\eta) + \delta & \text{si } \eta < x \leq \epsilon \\ 0 & \text{si } \epsilon < x \leq sup - \epsilon \\ -\gamma\frac{(sup-x-\eta)^2}{2(\epsilon-\eta} + \gamma(sup-x-\eta) + \delta & \text{si } sup - \epsilon < x \leq sup - \eta \\ \gamma\frac{(sup-x-eta)^2}{2\eta} + \gamma(sup-x-\eta) + \delta & \text{si } sup - \eta < x \leq sup \\ critMax & \text{si } sup < x \end{cases}$$

avec

$$\gamma = -2\frac{critMax}{\epsilon}$$

et

$$\delta = -\gamma\frac{(\epsilon-\eta)}{2}$$

La figure A.1 donne une idée des diverses formes obtenues pour la borne inf, en fonction des valeurs des deux paramètres ϵ et η. La forme est symétrique pour la partie droite, au voisinage de sup. Le paramètre ϵ définit la valeur de x pour laquelle la fonction donne zéro, tandis que η joue sur la forme de la courbe en fixant son point d'inflexion. Il est possible d'utiliser un ϵ et un η différent pour chaque partie de la courbe.

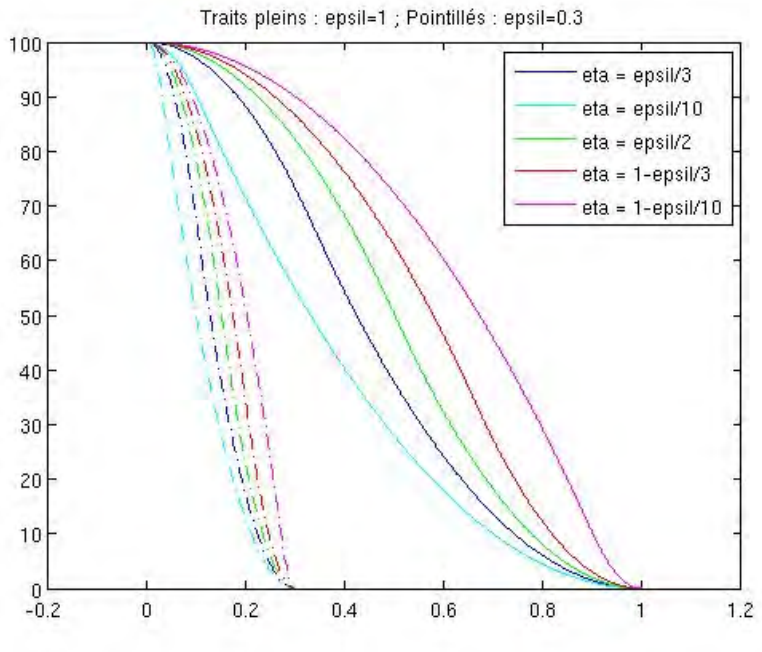

Figure A.1 – Partie gauche de fonctions barrières différemment paramétrées.